PROYECTO DE SISTEMAS INFORMÁTICOS

CURSO 2009/2010

DETECCIÓN DE PARES DE MOVIMIENTO PROPIO COMÚN MEDIANTE MINERÍA DE DATOS

Autores: Blanca Collado Iglesias

Antonio Javier Fernández Sánchez

Sara Pozuelo González

Director: Rafael Caballero Roldán

Autorización

Autorizamos a la Universidad Complutense de Madrid a difundir y utilizar con fines académicos, no comerciales, y mencionando expresamente a sus autores, tanto la propia memoria, como el código, la documentación y o el prototipo desarrollado.

Blanca Collado Iglesias

Antonio Fernández Sánchez

Sara Pozuelo González

Agradecimientos

A mi madre, por conseguir calmarme en los momentos clave, y a Álvaro, por sufrirme y darme ánimos durante todo este estresante curso.

Blanca Collado Iglesias

A todas aquellas personas que me han ayudado a lo largo de este difícil camino, especialmente a mi madre y a mi hermana por levantarme cada vez que tropezaba.

A Ricardo Peña y Antonio Navarro por su inagotable paciencia.

Antonio Fernández Sánchez

A mi madre, por inculcarme el valor del esfuerzo y del trabajo bien hecho y siempre confiar en mí y en mis posibilidades de lograr cualquier reto.

Sara Pozuelo González

Y por supuesto, y de parte de todos:

A Rafa Caballero, por ser tan accesible y cercano, y por convertir nuestras preocupaciones en una sonrisa.

PROYECTO DE SISTEMAS INFORMÁTICOS

CURSO 2009/2010

DETECCIÓN DE PARES DE MOVIMIENTO PROPIO COMÚN MEDIANTE MINERÍA DE DATOS

Autores: Blanca Collado Iglesias

Antonio Javier Fernández Sánchez

Sara Pozuelo González

Director: Rafael Caballero Roldán

ÍNDICE

ÍNDICE DE FIGURAS

RESUMEN

La astronomía se enfrenta actualmente al reto de procesar y asimilar la ingente cantidad de datos que son generados diariamente por los distintos observatorios y equipos de investigación. Esta tarea a menudo se torna tediosa o imposible. CPMP es un software especializado que, utilizando técnicas de almacenamiento heterogéneo y minería de datos, ha probado su utilidad para el descubrimiento de nuevos pares de estrellas con movimiento propio común (estrellas dobles), de forma simple, intuitiva y extremadamente veloz. La aplicación, implementada en Java mediante una arquitectura multicapa basada en el modelo vista controlador, trabaja con catálogos de estrellas siguiendo las especificaciones del servicio web para astronomía VizieR, los cuales serán persistidos con el motor de bases de datos relacional MySQL. El usuario tendrá a su disposición diversas opciones para gestionar los catálogos, como por ejemplo, la visualización de las distintas estrellas a través de la aplicación online Aladin. Sin embargo, la característica más importante, flexible y potente de esta herramienta es la que permite al usuario definir sus propios scripts para realizar operaciones de procesamiento y filtrado sobre los catálogos. Para el desarrollo de este proyecto se ha seguido una metodología de trabajo fundamentada en la ingeniería del software, lo que incluye la realización de un plan de proyecto, la obtención de los requisitos funcionales de la aplicación y una adecuada gestión del riesgo y de la calidad del software. Gracias al uso de la aplicación han sido descubiertos 83 nuevos pares de estrellas, lo que ha dado lugar a dos publicaciones en el ámbito de la astronomía.

PALABRAS CLAVE

Almacenamiento heterogéneo, astronomía, base de datos, catálogo, estrella doble, Java, minería de datos, movimiento propio, MySQL, script, VizieR.

ABSTRACT

Astronomy is currently facing the challenge of processing and assimilating enormous amount of data daily generated by observers and different research teams. This task often becomes tedious or impossible. CPMP is a special purpose software that, using datawarehouse and data mining techniques, has proven its utility for the discovery of new common proper motion pairs of stars (double stars), in a simple, intuitive and extremely fast way. The application, implemented in Java using a software architecture based on the model view controller, works with star catalogs following the specifications of the astronomy web service VizieR. These catalogues are persisted with the relational databases engine MySQL. The user has various options available for catalogue management, for example, the display of different stars through the online application Aladin. However, the most important feature of this flexible and powerful tool is the one that enable users to define their own scripts in order to execute processing and filtering operations on catalogues. For the development of this project, it has been used a software engineering approach, including the completion of a project plan, a software requirement specification and a risk and software quality management. As a result of the use of CPMP, 83 new pairs of stars have been discovered, given raise to two new publications in the astronomy field.

KEYWORDS

Astronomy, catalogue, database, data mining, datawarehouse, double star, Java, MySQL, proper motion, script, VizieR.

1. INTRODUCCIÓN

La informática, definida como el conjunto de conocimientos científicos y técnicos que hacen posible el tratamiento automático de la información, ha ido incrementando su alcance funcional durante las últimas décadas hasta el punto de acabar convirtiéndose en una herramienta totalmente indispensable en cualquier campo de investigación.

Ciencias como la biología, la química o la medicina fusionadas con la informática han producido una interesante mezcla de conocimientos que, pese a no ser novedosa, si está emergiendo en la actualidad con muchísima fuerza. Tareas que antes eran prácticamente irrealizables o necesitaban años para su consecución, son ahora llevadas a cabo en pocas semanas gracias a la transferencia tecnológica entre los diversos campos de investigación. Como prueba de estas palabras solo debemos constatar los innumerables programas de Master y de Doctorado que están apareciendo al respecto.

Sin embargo, la informática no solo ha permitido aumentar la productividad de los más avanzados centros de investigación, sino que también ha dotado a aquellos profesionales o aficionados que no tienen a su disposición estos costosos recursos, de herramientas lo suficientemente potentes como para generar una valiosa producción científica.

Nuestro sistema informático, al que de ahora en adelante nos referiremos como CPMP (Common Proper Motion Pairs) nace con la intención de situarse dentro de esta segunda definición, más concretamente, en el ámbito de la astronomía.

1.1 Minería de datos

Las técnicas de minería de datos han demostrado su eficacia en el descubrimiento de nuevos pares de estrellas con movimiento propio común, como se observa en "New Northern hemisphere common proper-motion pairs" [G04] y en el artículo "Finding New Common Proper-Motion Binaries by Data Mining" [C10].

En estos trabajos ha resultado evidente que este proceso de búsqueda era normalmente sistemático, con ligeras diferencias según el tipo de catálogo. Por lo tanto, parecía interesante desarrollar un software especializado que automatizara las diferentes fases de este proceso de minería de datos.

1. Descarga del total o una parte del catálogo "C", normalmente desde el servicio web online Vizier (Allende & Dambert 1999). Algunas veces, son necesarias porciones de catálogos auxiliares, como por ejemplo para completar la información sobre el espectro, la magnitud visual, etc.
2. Importar la información contenida en C a una base de datos relacional como Access o MySQL (y también el catálogo auxiliar en caso de existir)
3. Obtener el producto Cartesiano de la tabla consigo misma ($D = C \times C$), es decir, crear una tabla de pares.
4. Borrar de D todos los pares con una separación mayor que un número elegido arbitrariamente, por ejemplo, 100 segundos.
5. Borrar de D todos los pares que ya pertenecen al catálogo WDS (Washington Double Star Catalog, Mason et al., 2003).
6. Aplicar algún tipo de criterio de filtrado para conservar en D solo los posibles pares con movimiento propio común. Un criterio habitual para este tipo de filtrados es el criterio de Halbwachs. (Halbwachs, 1986).
7. Si es posible, aplicar más criterios que puedan ayudar a incrementar la calidad del resultado, por ejemplo, rechazando aquellos pares que no estén físicamente adyacentes.
8. Finalmente, contrastar cada par con las places fotográficas disponibles en Aladin, buscando dos estrellas con movimiento y la misma información astronómica en la posición esperada.
9. Completar el resultado con información astronómica apropiada extraída de los catálogos auxiliares.

El proceso de minería de datos

Existen, no obstante, una serie de problemas iniciales a la hora de gestionar y trabajar sobre inmensos catálogos de estrellas, como pueden ser el tamaño de estos y la falta de estándares para representarlos. CPMP proporciona al usuario una herramienta simple, intuitiva, automatizada y ante todo, extremadamente veloz, para gestionar cualquier tipo de catálogos de estrellas de forma unificada.

1.2 Common Proper Motion Pairs (CPMP)

CPMP es una aplicación implementada en Java mediante una arquitectura multicapa basada en el patrón MVC (Modelo-Vista-Controlador). Internamente está divido en dos módulos principales más un menú de ayuda al usuario.

El primero se trata de un sistema de gestión de catálogos de estrellas. Este modulo permitirá al usuario importar masivamente catálogos de estrellas a partir de ficheros binarios o de texto. Aunque todas las estrellas dentro de un catálogo tengan el mismo formato, dos catálogos diferentes no tienen porque seguir la misma representación. Es decir, todas las estrellas dentro de un mismo catálogo están representadas de la misma manera, pero probablemente este formato sea diferente al de las estrellas de otro catálogo. Los catálogos persistirán en la aplicación a través del motor de bases de datos relacional MySQL. A la hora de trabajar con conjuntos de estrellas nos referiremos siempre a estos como catálogos.

Una vez importado uno o varios catálogos, el usuario podrá gestionarlos por separado o conjuntamente, teniendo disponible por pantalla un listado de estos. Individualmente, tendrá la opción de renombrar un catálogo, borrarlo, exportarlo de nuevo a un fichero de texto, mostrar sus campos o mostrarlo por pantalla. Si el usuario ha elegido esta última opción, la aplicación le dará la posibilidad de mostrar una imagen de cada una de las estrellas incluidas en dicho catálogo, siempre y cuando hayan sido fotografiadas previamente y estén incluidas dentro de la bases de datos de Aladin, otro de los servicios web del Centre de Données Aastronomiques de Strasbourg.

Si el usuario desea realizar operaciones sobre más de un catálogo a la vez, tendrá la opción de unir dos catálogos siempre que sus estrellas estén igualmente

representadas dentro de la aplicación, o cruzarlos para encontrar pares de estrellas que cumplan unas determinadas características de proximidad.

El segundo modulo consiste en un entorno de trabajo, una IDE para el desarrollo de scripts. Estos scripts constituirán una manera potente, flexible y veloz de trabajar con los catálogos y realizar minería de datos sobre ellos. El usuario dispondrá en principio de 10 operaciones básicas, pero podrá construir a partir de ellas funciones más complejas que usar en los scripts. Tanto estas funciones como los scripts pueden ser guardados y utilizadas a posteriori. Información mucho más completa sobre el funcionamiento y las posibilidades de estos dos módulos, así como diversos ejemplos y un manual de uso pueden ser encontrados a lo largo de este documento.

1.3 Definiciones, acrónimos, y abreviaciones

- **Estrella doble:** La mayoría de las estrellas que podemos observar en el cielo no están aisladas sino que forman sistemas de dos estrellas. Aunque es prácticamente imposible detectar ambas estrellas a simple vista, a través de telescopios u otros métodos indirectos es posible desdoblar estos sistemas en sus componentes. A veces, esta proximidad de las dos estrellas se debe únicamente a un efecto de perspectiva, es decir, la vemos próximas en el espacio pero realmente se encuentran a mucha distancia una de la otra. A estos sistemas se los conoce como dobles ópticas, y carecen de importancia científica. En cambio, otros pares están tan cercanos entre sí que existe relación física entre sus estrellas, e incluso en algunos casos podemos observar que una de ellas se mueve alrededor de la otra respondiendo a las leyes de la Gravitación Universal y de Kepler sobre el movimiento orbital. En realidad, ambas estrellas se mueven alrededor del centro común de masa. A esta clase de pares con una relación gravitacional entre sus componentes se los conoce como estrellas binarias.
- **Estrella binaria visual:** Dentro de las binarias, son aquellas en las que ambos componentes están lo suficientemente separados como para ser

detectadas directamente con los sistemas ópticos. Si estés dobles muestran un indicio de movimiento orbital, se las denomina binarias orbitales. Si aunque no se haya determinado fehaciente un movimiento orbital, tienen en común el movimiento propio, se las llama pares con movimiento propio común (MPC).

El estudio de estas estrellas binarias es de fundamental importancia para la astrofísica puesto que son nuestra fuente directa de conocimiento de las masas estelares, parámetro fundamental para comprender la evolución de estos astros.

A pesar de su fácil observación, este tipo de binarias es el más complicado de detectar ya que sus períodos orbitales suelen ser del orden de cientos de años. Ni siquiera dos estrellas cercanas tendrían por qué ser binarias pues bien podrían ser dos estrellas que han cruzado ocasionalmente sus trayectorias. La prueba clave la dan siempre sus trayectorias respectivas. Para poder apreciar el movimiento mutuo de las binarias visuales hay que comparar sus coordenadas en el cielo durante años distintos, a veces incluso décadas de diferencia.

- **CPMP:** Pares de estrellas con movimiento propio común (Common Proper Motion Pairs). Es el caso en el que dos o más estrellas se mueven juntas a través del espacio, es decir, tienen el polo Norte del Sol, o la rotación de un cuerpo sobre su eje en el sentido de las agujas del reloj visto desde encima del polo Norte del Sol.
- **RA y DEC**: coordenadas de la esfera celeste se denominan declinación (abreviado DEC, o δ) y ascensión recta (abreviado RA o α). Ambas juntas reciben el nombre de coordenadas ecuatoriales porque se basan en una esfera cuyo ecuador y polos se corresponden con los de la Tierra. La declinación es como la latitud y se mide en grados, negativos si cae al sur del ecuador. La ascensión recta es como la longitud, pero se mide en horas (de 0 a 24).
- **RAJ2000 y DEJ2000:** La ascensión recta y la declinación de cada estrella experimentan cambios lentos a lo largo de los años debido a la precesión, un desplazamiento gradual de la dirección del eje terrestre.

Eso implica que la ascensión recta y la declinación carecen de sentido si se desconoce la época (la fecha e instante en que una observación astronómica fue realizada) a la que pertenecen. La mayoría de mapas y catálogos estelares actuales usan la época 2000, o para mayor precisión, J2000.0 (mediodía en UTC del 1 de enero de 2000, doce horas después del comienzo del año).

Muchos catálogos de posiciones planetarias, incluidos los anuarios astronómicos, usan el equinoccio de la fecha, es decir, la época corresponde a la fecha para la que se han hecho las predicciones.

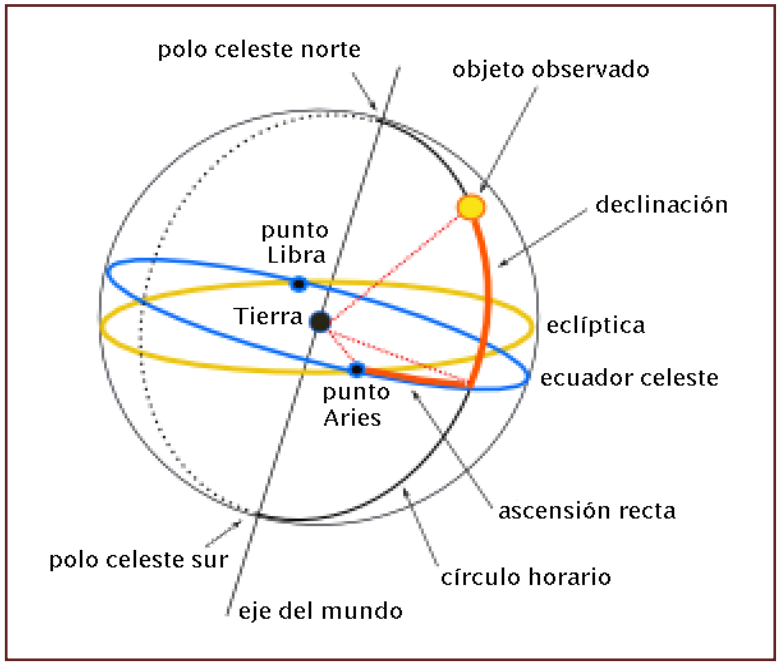


Figura 1 Esfera Celeste

◊ **Criterios de Halbwachs:** Estudiar la naturaleza de las estrellas dobles significa encontrar las condiciones mínimas necesarias para conocer la probabilidad de que la pareja sea óptica, de movimiento propio común (MPC) o de origen común o física.

Para este estudio nos basamos en la aplicación de los criterios utilizados por el astrónomo profesional J.L. Halbwachs. En el trabajo realizado por

Halbwachs se consideraba que dos estrellas tienen movimientos similares, las llamadas "estrellas de movimiento propio común (MPC)" si el nivel de semejanza es del 95% y satisfacen los siguientes criterios:

1. $(\mu 1 - \mu 2)\ 2 < -2\ (\sigma 12 + \sigma 22)\ \ln\ (0.05)$
2. $|\mu 1|, |\mu 2| \geq 50$ mas/yr
3. $\rho / |\mu 1|, \rho / |\mu 2| < 1000$ yr

Donde $\mu 1$, $\mu 2$ son los dos vectores de movimiento común, σi es el error medio de las proyecciones en los ejes de coordenadas de μi, y ρ es la separación angular de las dos estrellas.

- **VizieR:** servicio de catálogos astronómicos proporcionado por el Centre de Données astronomiques de Strasbourg (CDS).
- **Aladin:** Aladin es un atlas del orbe celeste interactivo que permite al usuario visualizar imágenes astronómicas digitalizadas, proporcionado por el Centre de Données astronomiques de Strasbourg (CDS). Estará integrado en la aplicación para poder mostrar al usuario las imágenes de las estrellas seleccionadas.
- **CDS:** Centre de Données astronomiques de Strasbourg.
- **SRS:** Especificación de requisitos software (Software Requirements Specification).
- **API:** Interfaz de programación de aplicaciones (Aplication Programming Interface).

2. OBJETIVO DEL PROYECTO

Los objetivos de este proyecto serán principalmente:

- Desarrollo de un sistema de gestión de catálogos de estrellas capaz de abstraerse de la representación interna de las mismas. Este sistema, aparte de las operaciones habituales en un gestor de información y otras más

específicas, debe incluir como funcionalidad principal la posibilidad de cruzar cualquier tipo de catálogo entre sí para encontrar pares de estrellas con movimiento propio común (candidatas a estrellas dobles). El interfaz gráfico del sistema debe ser simple e intuitivo.

- Pese al enorme tamaño de los catálogos (pueden llegar a pesar varios gigas), el sistema debe permitir tanto importar como exportar los mismos con la mayor velocidad posible. La funcionalidad de cruzar catálogos, crítica en el sistema, debe cumplir los mismos requisitos, por lo que serán imprescindibles el uso de algoritmos de divide y vencerás. Actualmente llevar a cabo estas tareas de forma manual y no optimizada conlleva semanas de trabajo.
- Desarrollo de un Interfaz de Desarrollo mediante el cual el usuario pueda escribir y ejecutar scripts. Estos scripts estarán compuestos por operaciones atómicas previamente definidas y por otras llamadas a funciones más complejas desarrolladas por el usuario. Estos scripts constituirán una potente herramienta para realizar cualquier tipo de labor de minería de datos sobre todo tipo de catálogos.
- Desarrollo de un interfaz con el usuario que le permita visualizar por pantalla placas fotográficas de las estrellas a través del servicio web **Aladin** con un solo click de ratón.
- Desarrollo de un menú de ayuda intuitivo y potente en inglés, el cual ofrecerá no solo instrucciones de cómo sacar el máximo rendimiento a la aplicación sino también nociones básicas de astronomía. El menú de ayuda deberá tener un buscador y estar indexado.

3. ESTRUCTURA DEL DOCUMENTO

La estructura de la presente memoria se expone a continuación:

- El Capítulo 1 presenta la Especificación de Requisitos Software, realizada siguiendo el estándar **ISO 830** del **IEEE**. En dicho capítulo se exponen de forma clara y concisa los servicios y restricciones de nuestro sistema software.

- El Capítulo 2 describe el Plan de Proyecto Software seguido durante el desarrollo de nuestro proyecto. En este apartado se recoge toda la información referente a la gestión del proyecto y que sirvió de guía durante el desarrollo del mismo.
- El Capítulo 3 detalla la arquitectura e implementación del proyecto. En este capítulo se describe con total claridad la metodología seguida para diseñar e implementar la aplicación.
- El Capítulo 4 muestra las principales conclusiones y resultados obtenidos con el desarrollo del proyecto. También se exponen algunas de las posibles líneas de investigación futuras.
- El apartado de **Apéndices** contiene los siguientes:
  - Apéndice 1: Contiene el manual de la aplicación.
  - Apéndice 2: Contiene un pequeño tutorial que explica cómo descargar un catálogo desde el servicio *online* de **VizieR** (servicio de catálogos astronómicos proporcionado por el Centre de Données astronomiques de Strasbourg (CDS)).
  - Apéndice 3: Incluye los dos artículos a los que el desarrollo de la aplicación ha dado lugar.
  - Apéndice 4: Muestra la Estructura de Descomposición del Trabajo utilizada durante la planificación del proyecto.
  - Apéndice 5: Expone la estimación del esfuerzo del proyecto.
  - Apéndice 6: Contiene el diagrama de Gantt.
  - Apéndice 7: Muestra la tabla de recursos del proyecto.
  - Apéndice 8: Incluye las actas de las reuniones mantenidas con el director del proyecto a lo largo del curso.
- Finalmente el apartado de Bibliografía expone las referencias que fueron consultadas durante el desarrollo del proyecto.

CAPÍTULO 1
ESPECIFICACIÓN DE REQUISITOS SOFTWARE

1.1 INTRODUCCIÓN

1.1.1 Propósito

El objetivo de la especificación de requisitos software (SRS) es describir de forma concisa los servicios y restricciones de nuestro sistema software. La SRS establece una base de acuerdo o contrato entre el cliente y los desarrolladores, que en nuestro caso particular, serán el profesor de la asignatura Sistemas Informáticos, incluida en la titulación de Ingeniería Informática de la Universidad Complutense de Madrid (Código UCM-360-98-434), y el grupo formado para desarrollar el proyecto. Además, proporciona una guía de la estructura del sistema, es decir, una descripción de las funcionalidades que implementará este. A partir de esta especificación podremos empezar a estimar y planificar el proyecto para, posteriormente, desarrollar la aplicación según las indicaciones incluidas en este documento. Además, servirá como índice de referencia de las funcionalidades del sistema.

Como referencia para la creación de este documento se ha seguido el estándar ISO 830 del IEEE para la especificación de requisitos software.

1.1.2 Alcance Funcional

El contenido de esta sección se puede encontrar en el apartado 1.2 de la "Introducción" de esta memoria.

1.1.3 Definiciones, acrónimos, y abreviaciones

El contenido de esta sección se puede encontrar en el apartado 1.3 de la "Introducción" de esta memoria.

1.1.4 Apreciación Global

El resto de la SRS está organizada siguiendo el siguiente esquema:

- **Descripción general:** describe los factores generales que afectan al producto y sus requisitos.
  - **Perspectiva del producto:** establece el contexto de implantación del producto y el uso del interfaz del sistema, interfaces de usuario, hardware, interfaces de comunicación, etc.
  - **Funciones del producto:** describe las funciones principales del software.
  - **Características de usuario:** describe el nivel educativo, experiencia y capacidad del usuario.
  - **Restricciones:** Políticas de regulación, limitaciones hardware e interfaces con otras aplicaciones, operaciones en paralelo, funciones de auditoría y de control, requisitos de fiabilidad, etc.
  - **Supuestos y dependencias:** Identifica los factores que afectan a los requisitos de la SRS.
  - **Requisitos futuros:** indicará los requisitos que se incluirán en versiones futuras del software.
- **Requisitos específicos:** Contienen todos los requisitos del sistema, con nivel de detalle mayor que el apartado anterior. Constituye la base del diseño que posteriormente será implementado.
  - **Interfaces:** descripción detallada de todas las entradas y salidas del sistema software.
  - **Funciones:** define las acciones fundamentales que tendrán lugar en el software durante la aceptación y procesamiento de la entrada y durante el procesamiento y generación de la salida.
  - **Requisitos de Rendimiento:** requisitos del rendimiento del sistema.
  - **Requisitos de la Base de datos lógica:** Requisitos lógicos de la información que residirá en la base de datos.

- **Restricciones de Diseño:** Restricciones impuestas al diseño por adecuación a otros estándares, limitaciones hardware, etc.
- **Atributos del sistema software:** Atributos del sistema software que pueden servir como requisitos.

1.2 DESCRIPCIÓN GENERAL

1.2.1 Perspectiva del producto

Esta aplicación ha sido desarrollada en el marco de la asignatura Sistemas Informáticos, de la titulación de Ingeniería Informática de la Universidad Complutense de Madrid (Código UCM-360-98-434). Nuestro sistema deberá ser capaz de gestionar de forma eficiente el procesamiento y gestión de inmensos catálogos de estrellas, habilitando de forma efectiva al usuario para realizar complejas operaciones de minería de datos.
Se tratará de una aplicación independiente y autónoma, por lo que no se considerarán dependencias funcionales con otros posibles sistemas informáticos que engloben al nuestro.

Deberá estar implementado de tal forma que se favorezca al máximo la compatibilidad, esto es, que pueda ser ejecutado en el mayor número de máquinas posibles independientemente de su Sistema Operativo, por lo que se usará Java como lenguaje de desarrollo debido a la compatibilidad que ofrece su máquina virtual. No se contempla que los distintos módulos del sistema informático puedan ser reutilizados para otras aplicaciones en el futuro, pese a lo cual, se buscará una implementación mediante una arquitectura software multicapa con alta cohesión y bajo acoplamiento.

Por último, la aplicación será monousuario, por lo que solo existirá un perfil administrador que tendrá acceso a todas las funcionalidades del sistema.

Posibles restricciones:

Interfaces del sistema: El sistema no formará parte de un sistema más grande.

Interfaces de usuario: El sistema consta de una interfaz gráfica que permitirá al usuario utilizar sus funcionalidades de una manera sencilla e intuitiva, así como hacer consultas del estado de los datos del sistema software.

Interfaces hardware: La comunicación entre el software y el hardware no requerirá de características especiales. Se buscará la máxima compatibilidad posible.

Interfaces software: La aplicación interaccionará de forma interna (esto es, oculta al usuario) con un software de persistencia de datos, y con la herramienta web "Aladin", que nos permitirá el acceso a las imágenes de las estrellas con las que trabajaremos.

- **Nombre Completo:** The Aladin Sky Atlas.
- **El número de la versión:** Release 6.
- **Referencias bibliográficas**: http://www.aanda.org/articles/aas/pdf/2000/07/ds1785.pdf
- **Fuente:** http://aladin.u-strasbg.fr/

Para la persistencia de datos, se utilizará en esta primera versión la aplicación de gestión de bases de datos MySql, aunque se contempla que el sistema permita en un futuro el trabajo con otros gestores.

- **Nombre Completo:** MySQL Community Server.
- **El número de la versión:** 5.1.44.
- **Referencias bibliográficas**: http://downloads.mysql.com/docs/refman-5.0-en.a4.pdf
- **Fuente:** http://www.mysql.com/downloads/mysql/

Interfaces de comunicación: No se requiere ninguna interfaz de comunicación.

Memoria: El sistema hardware que ejecute la aplicación deberá tener memoria suficiente para almacenar los datos de los que haga uso el programa, y posibilitar

el trabajo sobre ellos de forma veloz y efectiva. Los catálogos de estrellas pueden llegar a ocupar varios gigas de memoria.

Operaciones: Se tratará de una aplicación monousuario. Sólo será necesario un modo de operación, el cual permitirá utilizar cualquier funcionalidad del sistema.

Requisitos de adaptación: Para ejecutar el sistema y hacer uso de todas sus funcionalidades, el usuario deberá tener instalados previamente en su máquina un entorno de ejecución Java y el software de gestión de bases de datos, que en esta primera versión será MySql. Para alguna de las funcionalidades del sistema, como la de poder ver imágenes de las estrellas, se requerirá una conexión a Internet.

1.2.2 Funciones del producto

El sistema se descompone en dos módulos principales, a los que a partir de ahora nos referiremos como "Módulo de gestión de catálogos" y "Módulo de desarrollo de scripts":

a) Módulo de gestión de catálogos: Este modulo permitirá al usuario importar masivamente catálogos de estrellas a partir de ficheros binarios o de texto. Aunque todas las estrella dentro de un catálogo tengan el mismo formato, dos catálogos diferentes no tienen porque seguir la misma representación. Es decir, todas las estrellas dentro de un mismo catálogo están representadas de la misma manera, pero probablemente este formato sea diferente al de las estrellas de otro catálogo. Dichos catálogos de estrellas estarán agrupados de forma unificada dentro de la aplicación según las especificaciones de VizieR (servicio de catálogos astronómicos proporcionado por el Centre de Données astronomiques de Strasbourg (CDS)). En el marco de la aplicación, estos catálogos serán considerados tablas de la base de datos.

Una vez importado uno o varios catálogos, el usuario podrá gestionarlos por separado o conjuntamente, teniendo disponible por pantalla un listado de ellos. Individualmente, tendrá la opción de renombrar un catálogo, borrarlo, exportarlo de nuevo a un fichero de texto, ver los campos que lo componen o mostrarlo por

pantalla. Si el usuario ha elegido esta última opción, la aplicación le dará la posibilidad de mostrar una imagen de cada una de las estrellas importadas a la aplicación, siempre y cuando esté incluida dentro de Aladin, otro de los servicios web del CDS.

Si el usuario desea realizar operaciones sobre más de un catálogo a la vez, tendrá la opción de unir dos catálogos siempre que sus estrellas estén igualmente representadas dentro de la aplicación, o cruzarlos para encontrar pares de estrellas que cumplan unas determinadas características de proximidad.

Este módulo también permitirá al usuario resetear completamente toda la aplicación.

b) Módulo de desarrollo de scripts: Este módulo consistirá en un entorno de trabajo, una IDE, para el desarrollo y ejecución de scripts. Estos scripts constituirán una manera potente, flexible y veloz de trabajar con los catálogos y realizar minería de datos sobre ellos. El usuario dispondrá en principio de 10 operaciones básicas, pero podrá construir a partir de ellas funciones más complejas que usar en los scripts. Estas funciones podrán ser guardadas y utilizadas a posteriori. Este módulo también deberá hacer uso de ciertas funcionalidades muy básicas sobre los catálogos, como listarlos, borrarlos o mostrar sus campos, para facilitar el desarrollo de scripts. Es importante que este entorno de desarrollo sea lo más amigable posible con el usuario, y que proporcione una consola donde este pueda observar el resultado de las operaciones que ha realizado media scripts.

Las principales funciones de nuestro sistema, agrupadas en los módulos anteriormente descritos son:

Módulo de gestión de catálogos

- Importar catálogo VizieR.
- Crear cabecera VizieR.
- Exportar catálogo.
- Listar catálogos.
- Mostrar campos de un catálogo.

- Mostrar catálogo.
- Mostrar estrella.
- Renombrar catálogo.
- Borrar catálogo.
- Unir dos catálogos.
- Cruzar dos catálogos.
- Resetear aplicación.
- Dividir catálogos.

Módulo de desarrollo de scripts

- Escribir y ejecutar script.
- Escribir y guardar script.
- Cargar script.
- Escribir y guardar función.
- Cargar función.

1.2.3 Características de usuario

El usuario de esta aplicación deberá tener un conocimiento de astronomía de nivel medio para explotar todas las funcionalidades implementadas, además de entender el propósito y las posibilidades de la aplicación. Además, dicho usuario deberá estar familiarizado con el entorno de un sistema operativo con el cual poder ejecutar la aplicación. Por último, para algunas de las operaciones disponibles a la hora de realizar scripts, se necesitarán ciertas nociones muy básicas de lenguaje SQL, aunque el sistema incluirá un módulo de ayuda complementario que resultará más que suficiente para aprender a utilizar todas las operaciones del entorno de desarrollo de scripts.

1.2.4 Restricciones

Las restricciones impuestas a nuestro proyecto software son las descritas en la normativa de la asignatura Sistemas Informáticos de la Facultad de Informática de la Universidad Complutense de Madrid (Código UCM-360-98-434).

Dicha normativa puede encontrarse en el siguiente enlace: *http://www.fdi.ucm.es/SI/Normativa-SI-Junta05Nov07.pdf*

Aparte de dicha normativa general, para el curso académico 2009-2010 se han establecido las siguientes restricciones específicas para las fechas de entrega:

Entrega del borrador del proyecto al director del mismo: Viernes 11 de junio de 2010

Entrega de la memoria final en secretaría de alumnos: Viernes 2 de julio de 2010

Exposiciones públicas de los trabajos que no optan a MH: Del 5 al 8 de julio de 2010

Exposiciones públicas al tribunal de MH: Del 8 al 9 de julio de 2010

1.2.5 Supuestos y dependencias

Es bastante común en muchos proyectos que la especificación de los requisitos cambie a medida que avanza el desarrollo del mismo. Esto puede deberse a varios aspectos, entre los que puede estar el hecho de que los requisitos no hubieran sido definidos con exactitud en un principio. En nuestro caso, asumimos que nuestros requisitos son estables. En todo caso, se pueden producir pequeñas modificaciones o matices, las cuales quedarán reflejadas por escrito.

1.2.6 Requisitos futuros

Desde el primer momento se ha tenido en consideración la posibilidad de aumentar la funcionalidad de la aplicación. Por tanto, se desarrollará el sistema informático de forma que se minimice el impacto o las ramificaciones que pudiera causar la inclusión de nuevos módulos. Para tal efecto, se usarán técnicas de

Ingeniería del Software para reducir el acoplamiento dentro de la arquitectura software, buscando la máxima modularidad. Algunas de estas funcionalidades futuras pueden ser la inclusión de otros motores de persistencia de datos (XML, Oracle...), o la posibilidad de descargar los catálogos de Internet, una vez que el Centre de Données astronomiques de Strasbourg (CDS) libere un servicio web para tal propósito, y cuyo uso simplifique y acelere sensiblemente la obtención de dichos catálogos respecto a su obtención manual, cosa que actualmente no sucede.

1.3 REQUISITOS ESPECÍFICOS

1.3.1 Interfaces externas

Las entradas del sistema se harán por teclado, ratón y mediante uso de ficheros binarios o de texto presentes en la máquina del usuario. Los tipos de datos que almacenaremos en las entradas de las bases de datos serán **Varchar** , **Integer**, **Double** y **Bool**. La información se solicitará específicamente en cada función.

1.3.1.1 Módulo de gestión de catálogos

Nuestro sistema, y por tanto, las entradas a este, girarán entorno a catálogos de estrellas. El principal problema de estos catálogos a la hora de ser representados es la falta de un estándar que sirva de plantilla para su creación, por lo que cada catálogo suele tener distintos campos, es decir, no todos los catálogos describen las estrellas usando los mismos atributos.

Esto nos impide, a priori, definir un tipo o una estructura de datos para recoger de forma universal el tipo de datos "estrella", que nos permitiría a su vez utilizar capas de persistencia automatizadas como Hibernate.

Por tanto, la única solución posible es optar por representar dentro de la aplicación estas estrellas de forma dinámica. Es decir, solo cuando estemos importando los catálogos se podrá capturar la representación de las estrellas, y por ende, la de los catálogos que las recogen. Para ello, se hará uso de tres metatablas que se encontrarán en todo momento en la base de datos y que configurarán el "estado de la aplicación". Estas metatablas recogerán los nombres de los catálogos,

el número de estrellas que contienen y los campos utilizados para describirlas. Cada caso de uso de este sistema informático deberá actualizar estas metatablas, a las que de ahora en adelante nos referiremos como "estado de la aplicación". En el apartado 1.3.4 se ofrece una explicación más extensa de estos aspectos. Por tanto, un catálogo no estará representado por un tipo único, sino por una serie de tipos relacionados entre sí:

TCatalogo	TCatalogoFilas	TField
String UserName	String UserName	String name
String LocalName	String LocalName	String type
String Title	String Title	int column
	Long numFilas	String nombreTabla

TCatalogo: Definirá los identificadores de un catálogo, esto es, el nombre elegido por el usuario (UserName) para éste dentro de la aplicación, el identificador del catálogo dentro de VizieR (LocalName) y el título del catálogo (Title), que consiste en una pequeña descripción de este junto con su verdadero nombre (Ej: PPMX, UCAC3...)

TCatalogoFilas: Tendrá los mismos campos que su tipo padre, TCatálogos, pero añadirá el número de estrellas del catálogo, lo que nos será útil en ciertos casos de uso.

TField: Representará cada uno de los campos que describen a una estrella. Tendrá un nombre de campo, su tipo (Varchar, Integer, Double o Bool), la columna que le corresponde dentro de un catálogo y el nombre del catálogo que lo contiene.

En resumen, un catálogo estará representado dentro de la aplicación mediante el tipo TCatalogo más una lista de TFields.

El usuario deberá proveer los catálogos en formato texto, y podrá hacerlo de dos maneras, usando un catálogo con una cabecera de VizieR, o sin esta. Una

cabecera "VizieR" está compuesta por una plantilla que enumera los campos que describen a las estrellas del catálogo, y los tipos de dichos campos. Además, incluye la información sobre los identificadores del catálogo. Si el usuario usa catálogos que no llevan cabecera, deberá proporcionar a la aplicación previamente esta información.

Ejemplo de cabecera VizieR:

Identificadores

```
#
# VizieR Astronomical Server: vizier.u-strasbg.fr 2009-10-10T17:37:34
#   (replaces the 'Astrores' format originally described at
#    http://vizier.u-strasbg.fr/doc/astrores.htx)
#   In case of problem, please report to: question@simbad.u-strasbg.fr
#
#
#Coosys     J2000:      eq_FK5 J2000
#INFO -ref=VIZ4ad0b4124a37
#INFO -out.max=50000

#RESOURCE=yCat_1312
#Name: I/312
#Title: PPMX Catalog of positions and proper motions (Roeser+ 2008)
#Coosys     J2000_2000.000:   eq_FK5 J2000
#Table      I_312_sample:
#Name: I/312/sample
#Title: The PPMX Catalog (Position and Proper Motions eXtended)
```

Campos y sus tipos

```
#Column     _RAJ2000    (F9.5)      Right ascension (FK5)
Equinox=J2000.0 Epoch=J2000.000, proper motions taken into account
(computed by VizieR, not part of the original data)
      [ucd=pos.eq.ra;meta.main]
#Column     _DEJ2000    (F9.5)      Declination (FK5) Equinox=J2000.0
Epoch=J2000.000, proper motions taken into account  (computed by
VizieR, not part of the original data)    [ucd=pos.eq.dec;meta.main]
#Column     PPMX  (a16) Name (IAU convention HHMMSS.S+DDMMSSa) (3)
      [ucd=meta.id;meta.main]
#Column     RAJ2000     (F10.6)     Right Ascension J2000.0, epoch
2000.0      [ucd=pos.eq.ra;meta.main]
#Column     DEJ2000     (F10.6)     Declination J2000.0, epoch 2000.0
      [ucd=pos.eq.dec;meta.main]
#Column     pmRA  (F9.2)      Proper Motion in RA*cos(DEmas)
      [ucd=pos.pm;pos.eq.ra]
#Column     pmDE  (F9.2)      Proper Motion in DE
      [ucd=pos.pm;pos.eq.dec]
```

Campos y unidad que representan

```
_RAJ2000;_DEJ2000;PPMX;RAJ2000;DEJ2000;pmRA;pmDE
deg;deg;;deg;deg;mas/yr;mas/yr
```

Línea separadora

```
---------;---------;----------------;----------;----------;---------
```

Todo catálogo que deseemos insertar deberá contener los campos RAJ2000 y DEJ2000.

Las estrellas dentro de los catálogos deberán estar representadas en consonancia con el contenido de la cabecera. El valor de cada campo estará separado por un ";" del valor del campo siguiente. Todos los campos, salvo RAJ2000 y DEJ2000, pueden no tener asignado un valor. A continuación, ponemos como ejemplo 4 estrellas codificadas según la representación de la cabecera anterior:

```
20.98323;28.16028;012355.9+280937 ;20.983231;28.160278;-
281.4;-248.55
20.99546;25.473;012358.9+252822
;20.995456;25.472995;89.67;-74.28
21.08169;24.057;012419.6+240325 ;21.081687;24.056999;-
22.43;-117.11
21.09304;24.9457;012422.3+245644
;21.093042;24.945703;36.19;-96.83
```

1.3.1.2 Módulo de gestión de scripts

Para el módulo de desarrollo de scripts, el usuario tendrá 3 opciones principales de interactuar con los interfaces gráficos de la aplicación.

a) Crear un script

Pasamos a ofrecer las operaciones básicas con las que podemos crear, ya sea un script o una función con operaciones más avanzadas, junto a su notación correspondiente.

Operaciones:

Unir dos tablas	
Sintaxis	NombreTablaNueva <- merge(tabla1,tabla2);
Requerimientos	Las tablas con los mismos atributos y del mismo tipo.
Salida	La unión de catálogos sin repeticiones RAJ2000, DEJ2000

Renombrar	
Sintaxis	NombreTablaNueva <- rename(tabla);
Requerimientos	“tabla” debe existir en el sistema.
Salida	Nuevo catálogo clonado completamente a partir de “tabla”

Cruzar	
Sintaxis	NombreTablaNueva <- join(tabla1,tabla2)[segundos];
Requerimientos	“tabla1”, “tabla2” deben ser tablas simples, es decir, no tablas de pares.
Salida	Tabla de pares de entradas, con |dec1-dec2|<segundos & |ra1-ra2|* COS(|dec1+dec2 / 2|)<segundos.

Filtro	
Sintaxis	NombreTablaNueva <- filter(tabla)[condicionSQL];
Requerimientos	“tabla” debe existir en el sistema. “condiciónSQL” debe ser una condición SQL válida.
Salida	Nuevo catálogo con las estrellas de “tabla” menos aquellas filtradas según la condición especificada.

Distancia	
Sintaxis	NombreTablaNueva <- distance(tabla)[segundos];
Requerimientos	“tabla” es una tabla de pares existente en el sistema.
Salida	Elimina de “tabla” las parejas que están a más de esa distancia.

Campo nuevo	
Sintaxis	NombreTablaNueva <- newAttribute(tabla,AtributeName,AttributeType);
Requerimientos	“tabla” existe en el sistema. “AttributeType” debe de pertenecer al rango de tipos válidos definidos en la sección 3.1. “tabla” debe existir en el sistema.
Salida	Añade un atributo con el nombre y el tipo especificado a la tabla.

Dar valor a campo nuevo

Sintaxis	NombreTablaNueva <- attribute(tabla,AttributeName,Value)[Condicion SQL];
Requerimientos	"Value" es una expresión válida para cada fila de la tabla y debe estar entre comillas simples. CondiciónSQL debe ser una condición SQL válida. "tabla" debe existir en el sistema.
Salida	Crea un nuevo catálogo donde se da valor al campo "AttributeName" de todas las estrellas de "tabla" que cumplan la condición especificada.

Minus	
Sintaxis	TablaNueva <- Minus(Tabla1,Tabla2) ;
Requerimientos	"Tabla1" y "tabla2" deben existir en el sistema.
Salida	Borrar de "Tabla1" la entradas cuyo (RAJ2000, DEJ2000) está en "tabla2".

Borrar una tabla	
Sintaxis	TablaNueva <- delete(Tabla) ;
Requerimientos	"Tabla" debe existir en el sistema.
Salida	Borrar de la aplicación la tabla "Tabla". "TablaNueva" nunca se crea .

Borrar campo	
Sintaxis	TablaNueva <- deleteAttribute(Tabla,Campo);

Requerimientos	"Tabla" y "Campo" deben existir en el sistema. "Campo" debe ser un campo de "Tabla" distinto de RAJ2000 y DEJ2000.
Salida	Borrar de Tabla el atributo de nombre campo

Cada operación debe estar en su propia linea, y el script en su conjunto debe estar precedido por la palabra **begin** y finalizado por la palabra **end**.

Ejemplo de script:

```
BEGIN
 joinVmag1y2 <-join(PPMX_pmDE1_Vmag,PPMX_pmDE2_Vmag)[3600];
 joinVmag3y4 <-join(PPMX_pmDE3_Vmag,PPMX_pmDE3_Vmag)[3600];
 mergeJoinsVmag <- merge(joinVmag1y2 , joinVmag3y4 );
 filterVmagDeRA<- filter(joinVmag1y2)[RAJ2000<1.0];
 distanceVmag3y4<- distance(joinVmag3y4)[0.005];
 setFilterPM(asu1,asu2);
 minusJoinsVmag  <-minus(joinVmag1y2 , joinVmag3y4 );
 newAtMinusJoinsVmag<-newAttribute(minusJoinsVmag,campo,double);
 atrbDeMinusJoin<-attribute(newAtrbminusJoinsVmag,campo,'2.0')[RAJ2000>0];
END
```

Dentro de un script se pueden incluir llamadas a funciones que hemos creado con anterioridad. Crear dichas funciones será el contenido del siguiente apartado, pero primero, para hacer uso de una de estas funciones, hemos incluido la siguiente instrucción dentro del cuerpo del script:

setFilterPM(asu1,asu2);

setFilterPM hará referencia al nombre de la función y, los parámetros, a las tablas que hacen falta como parámetros para usar esa función.

b) Crear una función

Una función tiene una notación parecida a un script, por lo que es mejor ir directamente a las diferencias a partir de un ejemplo:

```
function setFilterPM(2)
begin
  temp1<-newAttribute($1,mu,double);
  temp2<-newAttribute(temp1,b_mu,double);
  temp3<-attribute(temp2,mu,'sqrt(pmra*pmra+pmde*pmde)')[true];
  temp4< attribute(temp3,b_mu,'sqrt(b_pmra*b_pmra+b_pmde*b_pmde)')[true];
  $2<-filter(temp4)[mu>=50 and b_mu>=50];
end
```

Para empezar, vemos que las funciones empiezan con una cabecera:

function setFilterPM(2)

- **function**: identificador que indica que el código coresponde a una función.
- **setFilterPM**: nombre de la función.
- **(2)**: número de tablas/entradas que va a necesitar.

Y continúan con un cuerpo:

```
begin
  temp1<-newAttribute($1,mu,double);
  temp2<-newAttribute(temp1,b_mu,double);
  temp3<-attribute(temp2,mu,'sqrt(pmra*pmra+pmde*pmde)')[true];
  temp4<-attribute(temp3,b_mu,'sqrt(b_pmra*b_pmra+b_pmde*b_pmde)')[true];
  $2<-filter(temp4)[mu>=50 and b_mu>=50];
end
```

Esto es prácticamente un script, pero vemos, que en lugar de hacer referencia a tablas, usamos el símbolo del dólar junto a un número. Si la función necesita 4 entradas o parámetros, $1-$2 harán referencia a ellos.

Supongamos que dentro de un script realizamos la llamada:

nueva(tabla1,tabla2,tabla3,tabla4);

En nuestro disco, tenemos el fichero de texto "nueva.f" que contiene:

```
function nueva(4)
begin
        $3<-merge($1,$2);
        $4<-filter($3)[Vmag>0];
end
```

A la hora de ejecutar ese script, dicha llamada hará que se ejecuten las siguientes instrucciones:

tabla3<-merge(tabla1,tabla2);

tabla4<-filter(tabla3)[Vmag>0];

c) Cargar/guardar un script o una función

Para su posterior uso, podemos guardar tanto scripts como funciones. La única restricción es que las funciones deberán tener el formato ***".f"***. En nuestro ejemplo anterior, guardaríamos esa función como "nueva.f". Los scripts no tienen ninguna restricción de tipo.

1.3.2 Funciones

Los siguientes requisitos funcionales definen las acciones fundamentales que deben tener lugar en el software, aceptando y procesando las entradas y procesando y generando las salidas.

1.3.2.1 Módulo de gestión de catálogos

a) Importar catálogo VizieR

- **Función:** Importar catálogo VizieR.
- **Prioridad:** Alta.
- **Estabilidad:** Alta.

- **Descripción:** Inserta un nuevo catálogo dentro de la base de datos de la aplicación, a partir de un fichero binario o de texto existente en la máquina del usuario.
- **Entrada:** Selección de un catálogo ya existente mediante ratón, nombre del nuevo catálogo por teclado. El catálogo debe contener una cabecera VizieR válida acorde a lo descrito en el apartado 1.3.1.1 de esta SRS.
- **Salida:** Se proveerá al usuario por pantalla de un resumen del resultado de la operación, ofreciendo el tiempo tardado en la ejecución, las estrellas insertadas, y el número de ellas duplicadas dentro del catálogo. Además. se producirá un refresco del listado de catálogos del sistema que se esté mostrando por pantalla. En caso de que se esté ofreciendo más de un listado (los dos módulos del sistema están visibles al usuario al mismo tiempo), ambos deberán actualizarse.
- **Origen:** Usuario del sistema.
- **Destino:** Sistema.
- **Necesita:** Fichero binario o de texto con una cabecera VizieR válida.
- **Acción:** Insertar una nuevo catálogo en la base de datos. Las estrellas duplicadas no son insertadas.
- **Precondición:** El nombre de la nueva tabla no debe existir previamente en el sistema.
- **Postcondición:** El estado del sistema queda actualizado. La nueva tabla es añadida con el catálogo correspondiente.
- **Efectos laterales:** -

b) Descargar catálogo VizieR

- **Función:** Descargar catálogo VizieR.
- **Prioridad:** Media.
- **Estabilidad:** Baja
- **Descripción:** Descarga un catálogo de estrellas mediante el servicio web online de VizieR

- **Entrada:** Selección de un catálogo disponible para descarga mediante click de ratón. Introducción de ciertos parámetros mediante teclado como formato de grados de las coordenadas de las estrellas
- **Salida:** Nuevo catálogo con una cabecera VizieR válida en forma de fichero de texto descargado en el disco del usuario.
- **Origen:** Usuario del sistema.
- **Destino:** Sistema.
- **Necesita:** Conexión a internet.
- **Acción:** Descarga un catálogo de estrellas mediante el servicio web online de VizieR. El usuario tendrá disponible un interfaz para interactuar con el servicio web.
- **Precondición:** Ninguna
- **Postcondición:** Ninguna
- **Efectos laterales:** -

c) Renombrar catálogo

- **Función:** Renombrar catálogo.
- **Prioridad:** Media.
- **Estabilidad:** Alta.
- **Descripción:** Crea un nuevo catálogo clonando uno ya existente. El usuario introducirá mediante teclado el nombre con el que persistirá este catálogo dentro de la aplicación.
- **Entrada:** Selección de un catálogo ya existente mediante ratón, nombre del nuevo catálogo por teclado.
- **Salida:** La salida de este caso de uso se tratará de un refresco del listado de catálogos del sistema que se esté mostrando por pantalla. En caso de que se esté ofreciendo más de un listado (los dos módulos del sistema están visibles al usuario al mismo tiempo), ambos listados deberán actualizarse.
- **Origen:** Usuario del sistema.
- **Destino:** Sistema.
- **Necesita:** Tabla con el listado de catálogos de la base de datos.

- **Acción:** Insertar un nuevo catálogo en la base de datos clonando completamente uno ya existente.
- **Precondición:** El nombre de la nueva tabla no debe existir previamente en el sistema.
- **Postcondición:** Actualización de las tablas de la base de datos que guardan el estado del sistema. Añadida una nueva tabla con el catálogo correspondiente.
- **Efectos laterales:** -

d) Exportar catálogo

- **Función:** Exportar catálogo.
- **Prioridad:** Alta.
- **Estabilidad:** Alta.
- **Descripción:** Exporta un catálogo existente en la base de datos a un fichero de texto en la máquina del usuario. Este catálogo empezará por una cabecera VizieR válida generada por la aplicación.
- **Entrada:** Selección de un catálogo ya existente mediante ratón, nombre del nuevo catálogo por teclado. El usuario deberá insertar por ratón/teclado los campos, y los tipos de estos.
- **Salida:** Nuevo catálogo con una cabecera VizieR válida en forma de fichero de texto
- **Origen:** Usuario del sistema.
- **Destino:** Sistema.
- **Necesita:** Tabla con el listado de catálogos de la base de datos.
- **Acción:** Exporta un catálogo de la base de datos.
- **Precondición:** Debe existir al menos un catálogo en la base de datos.
- **Postcondición:** Fichero generado en la máquina del usuario.
- **Efectos laterales:** El usuario puede no tener suficiente espacio en disco.

e) Listar catálogos

- **Función:** Listar catálogos.

- **Prioridad:** Alta.
- **Estabilidad:** Alta.
- **Descripción:** Listar los catálogos presentes dentro de la aplicación por pantalla.
- **Entrada:** Petición de listar los catálogos por ratón. Esto se producirá también de forma indirecta cuando el usuario demande un servicio de la aplicación que necesite mostrar al usuario una lista de catálogos.
- **Salida:** lista de catálogos por pantalla.
- **Origen:** Usuario del sistema / sistema.
- **Destino:** Sistema.
- **Necesita:** Tabla con el listado de catálogos de la base de datos.
- **Acción:** Listar catálogos.
- **Precondición:** Ninguna, esta función puede ser llamada aún sin haber catálogos insertados.
- **Postcondición:** La base de datos no se ve afectada.
- **Efectos laterales:** -

f) Mostrar campos de un catálogo

- **Función:** Mostrar campos de un catálogo.
- **Prioridad:** Baja.
- **Estabilidad:** Alta.
- **Descripción:** Mostrar los campos de un catálogo por pantalla, junto con su tipo.
- **Entrada:** Petición de listar los campos por ratón.
- **Salida:** Lista de campos por pantalla.
- **Origen:** Usuario del sistema.
- **Destino:** Sistema.
- **Necesita:** Tabla con el listado de catálogos de la base de datos. Tabla con los campos de cada catálogo.

- **Acción:** Tras la petición del usuario, el sistema accederá a la base de datos donde se encuentran almacenados los campos de cada catálogo y mostrará los campos del catálogo solicitado.
- **Precondición:** Al menos un catálogo insertado en la BBDD.
- **Postcondición:** El estado del sistema no se ve afectado.
- **Efectos laterales:** -

g) Mostrar un catálogo

- **Función:** Mostrar un catálogo.
- **Prioridad:** Alta.
- **Estabilidad:** Alta.
- **Descripción:** Mostrar todas las estrellas de un catálogo por pantalla.
- **Entrada:** Petición de mostrar un catálogo mediante click de ratón.
- **Salida:** Lista con las estrellas del catálogo solicitado.
- **Origen:** Usuario del sistema.
- **Destino:** Sistema.
- **Necesita:** Tabla con el listado de catálogos de la base de datos. Tabla con los campos de cada catálogo. Tabla con el catálogo solicitado.
- **Acción:** Tras la petición del usuario, el sistema accederá a la base de datos donde se encuentra almacenado el catálogo, y lo mostrará por pantalla en intervalos si el número de estrellas en el catálogo es demasiado para una única tabla. Estos intervalos quedaran definidos por un número de estrellas máximo que se pueden mostrar al mismo tiempo, y se representarán en grados.
- **Precondición:** Al menos un catálogo insertado en la BBDD.
- **Postcondición:** El estado del sistema no se ve afectado
- **Efectos laterales:** -

h) Mostrar una estrella

- **Función:** Mostrar una estrella.
- **Prioridad:** Media.

- **Estabilidad:** Baja.
- **Descripción:** Mostrar la imagen de la estrella seleccionada por pantalla.
- **Entrada:** Petición de mostrar una estrella mediante click de ratón.
- **Salida:** Imagen de la estrella solicitada.
- **Origen:** Usuario del sistema.
- **Destino:** Sistema.
- **Necesita:** Este caso de uso necesita que la imagen demandada se encuentre disponible online en Aladin.
- **Acción:** Este caso de uso solo podrá suceder cuando se esté mostrando un listado de estrellas por pantallas. En ese momento, al hacer click de ratón sobre una de estas estrellas se accederá al servicio web del CDS llamado Aladin para mostrar al usuario la imagen correspondiente.
- **Precondición:** Al menos un catálogo insertado en la BBDD. El catálogo debe tener al menos una estrella.
- **Postcondición:** El estado del sistema no se ve afectado.
- **Efectos laterales:** -

i) Borrar un catálogo

- **Función:** Borrar un catálogo.
- **Prioridad:** Media.
- **Estabilidad:** Alta.
- **Descripción:** Borrar un catálogo de la base de datos.
- **Entrada:** Petición de borra un catálogo mediante click de ratón.
- **Salida:** Confirmación de catálogo borrado. Además se producirá un refresco del listado de catálogos del sistema que se esté mostrando por pantalla. En caso de que se esté ofreciendo más de un listado (los dos módulos del sistema están visibles al usuario al mismo tiempo), ambos listados deberán actualizarse.
- **Origen:** Usuario del sistema.
- **Destino:** Sistema.

- **Necesita:** Tabla con el listado de catálogos de la base de datos.
- **Acción:** El usuario solicitará borrar un catálogo mediante click de ratón, y el sistema accederá a la base de datos para eliminarlo de esta.
- **Precondición:** Al menos un catálogo añadido a la BBDD.
- **Postcondición:** El estado del sistema se ve actualizado con la eliminación de este catálogo y las referencias existentes a él.
- **Efectos laterales:** -

j) Unir dos catálogos

- **Función:** Unir dos catálogos.
- **Prioridad**: Media.
- **Estabilidad**: Media.
- **Descripción**: Este caso de uso unirá dos catálogos en uno nuevo. Ya que normalmente los catálogos se suelen dividir a la hora de descargarlos debido a su tamaño, es bastante útil poder unirlos de nuevo dentro de la base de datos.
- **Entrada:** Petición de unir dos catálogos mediante click de ratón. Ambos catálogos deberán tener el mismo formato.
- **Salida:** Confirmación de nuevo catálogo añadido. Además se producirá un refresco del listado de catálogos del sistema que se esté mostrando por pantalla. En caso de que se esté ofreciendo más de un listado (los dos módulos del sistema están visibles al usuario al mismo tiempo), ambos listados deberán actualizarse.
- **Origen:** Usuario del sistema.
- **Destino:** Sistema.
- **Necesita**: Tabla con el listado de catálogos de la base de datos.
- **Acción:** El usuario solicitará unir dos catálogos mediante click de ratón. El sistema comprobará si los catálogos tienen el mismo formato y procederá a unirlos en un nuevo catálogo. No se considerarán estrellas duplicadas.
- **Precondición:** Al menos dos catálogo añadidos a la BBDD con el mismo formato.

- **Postcondición:** El estado del sistema se ve actualizado con la adición del nuevo catálogo y las consiguientes referencias a este dentro de las metatablas.
- **Efectos laterales:** Elevado uso de espacio en disco al duplicarse los dos catálogos de origen.

k) Cruzar dos catálogos

- **Función:** Cruzar dos catálogos.
- **Prioridad**: Alta.
- **Estabilidad**: Alta.
- **Descripción**: Se cruzarán dos catálogos que pueden tener distinto formato. Se podrá cruzar un catálogo consigo mismo. Cruzaremos los catálogos respecto a una determinada distancia cuantificada en segundos, es decir, el resultado de esta operación será un nuevo listado de estrellas o catálogo en el sistema, que incluya aquellos pares de estrellas cuya distancia, medida en segundos, sea menor que la estipulada por el usuario. Para calcular esta distancia entre estrellas utilizaremos una aproximación de los criterios de Halbwach [H86].
- **Entrada:** Petición de cruzar dos catálogos mediante click de ratón. Distancia cuantificada en segundos mediante teclado.
- **Salida:** Confirmación de operación realizada mostrando el número de pares encontrados. Además se producirá un refresco del listado de catálogos del sistema que se esté mostrando por pantalla. En caso de que se esté ofreciendo más de un listado (los dos módulos del sistema están visibles al usuario al mismo tiempo), ambos listados deberán actualizarse.
- **Origen:** Usuario del sistema.
- **Destino:** Sistema.
- **Necesita**: Tabla con el listado de catálogos de la base de datos.
- **Acción:** El usuario solicitará cruzar dos catálogos mediante click de ratón e introducirá la distancia máxima a la que desea se encuentren los posibles pares. El sistema producirá un producto cartesiano de ambos catálogos

para poder seleccionar aquellos pares que estén dentro del criterio de búsqueda. Los algoritmos usados para realizar este proceso deben tener como uno de los objetivos principales la eficiencia dado lo costoso de la operación.

- ◊ **Precondición:** Al menos dos catálogo añadidos a la BBDD.
- ◊ **Postcondición:** El estado del sistema se ve actualizado con la adición del nuevo catálogo y las consiguientes referencias a este dentro de las metatablas.
- ◊ **Efectos laterales:** Elevado uso de memoria y alta carga del sistema.

l) Resetear aplicación

- ◊ **Función:** Resetear aplicación.
- ◊ **Prioridad**: Media.
- ◊ **Estabilidad**: Alta.
- ◊ **Descripción**: Resetea completamente el estado del sistema a su configuración inicial, es decir, previa a cualquier uso. Para ello, borra cualquier catálogo añadido y vacía el contenido de las metatablas.
- ◊ **Entrada**: Click de ratón.
- ◊ **Salida**: La salida de este caso de uso se tratará de un refresco del listado de catálogos del sistema que se esté mostrando por pantalla. En caso de que se esté ofreciendo más de un listado (los dos módulos del sistema están visibles al usuario al mismo tiempo), ambos listados deberán actualizarse.
- ◊ **Origen**: Usuario del sistema.
- ◊ **Destino**: Sistema.
- ◊ **Necesita**: Este caso de uso no necesita específicamente nada para producirse.
- ◊ **Acción**: Resetea la aplicación borrando todos los catálogos de la base de datos y vaciando las metatablas.
- ◊ **Precondición**: -
- ◊ **Postcondición**: El estado del sistema volverá a su configuración inicial, es decir, previa a cualquier uso.

- **Efectos laterales**: Al borrarse todos los catálogos, es bastante probable que los scripts realizados hasta el momento dejen de ser útiles.

m) Dividir catálogos

- **Función:** Dividir catálogos.
- **Prioridad**: Baja.
- **Estabilidad**: Alta.
- **Descripción**: Divide un catálogo de pares, es decir, aquellos producidos al cruzarse dos catálogos, en dos tablas independientes cada una con uno de los miembros de cada par.
- **Entrada**: Selección de catálogos mediante click de ratón.
- **Salida**: Confirmación de la operación. Además, se producirá un refresco del listado de catálogos del sistema que se esté mostrando por pantalla. En caso de que se esté ofreciendo más de un listado (los dos módulos del sistema están visibles al usuario al mismo tiempo), ambos listados deberán actualizarse.
- **Origen**: Usuario del sistema.
- **Destino**: Sistema.
- **Necesita**: Listado de catálogo de pares mostrándose por pantalla.
- **Acción**: Crea dos nuevas tablas en la base de datos con los catálogos generados al dividir el catálogo origen en dos.
- **Precondición**: Debe existir al menos un catálogo de pares en el sistema.
- **Postcondición**: El estado del sistema se ve actualizado con la adición de los nuevos catálogos y las consiguientes referencias a este dentro de las metatablas. El catálogo origen no es borrado de la aplicación.
- **Efectos laterales**: -

1.3.2.2 Módulo de desarrollo de scripts

a) Escribir y ejecutar script

- **Función:** Escribir y ejecutar script.

- **Prioridad:** Alta.
- **Estabilidad:** Alta.
- **Descripción:** Escribir un nuevo script acorde a las especificaciones incluidas en el apartado 1.3.1.2 de esta SRS. Posteriormente, el usuario podrá ejecutar este script.
- **Entrada:** El usuario escribirá un script por teclado sobre una interfaz gráfica diseñada para el caso. Ejecución del script mediante click de ratón.
- **Salida:** La salida de este caso de uso se producirá una vez el usuario haya decidido ejecutar el script. Se mostrará por pantalla una consola en la cual se desglosará el resultado de cada operación, o los errores en caso de que hayan existido. Además, como la mayoría de las operaciones del script generan nuevos catálogos, se producirá un refresco del listado de catálogos del sistema que se esté mostrando por pantalla. En caso de que se esté ofreciendo más de un listado (los dos módulos del sistema están visibles al usuario al mismo tiempo), ambos listados deberán actualizarse.
- **Origen:** Usuario del sistema.
- **Destino:** Sistema.
- **Necesita:** Los scripts harán referencia a catálogos dentro del sistema, por lo que será necesario mostrar al usuario por pantalla en todo momento un listado de catálogos.
- **Acción:** El usuario escribirá un script acorde a las especificaciones del apartado 1.3.1.2. Una vez ejecute el script, el sistema parseará y compilará las instrucciones de este script, verificará su corrección sintáctica y semántica y procesará las llamadas a funciones externas que se hayan incluido. Posteriormente, ejecutará secuencialmente cada una de las instrucciones, mostrando por la consola su resultado. En caso de producirse cualquier error, se parará la ejecución del script guardándose el resultado de las instrucciones ya finalizadas.
- **Precondición:** Debe existir al menos una tabla en el listado de catálogos para que se pueda realizar alguna de las posibles operaciones. Cada una de

estas operaciones tiene sus propias restricciones, definidas en el apartado 1.3.1.2 de esta SRS.

- **Postcondición:** El estado del sistema se ve actualizado con la adición o eliminación de nuevos catálogos y las consiguientes referencias a estos dentro de las metatablas.
- **Efectos laterales:** Algunas de las operaciones posibles del script pueden cargar mucho al sistema y su resultado ocupar bastante espacio en disco.

b) Escribir y guardar script

- **Función:** Escribir y guardar script.
- **Prioridad:** Alta.
- **Estabilidad:** Alta.
- **Descripción:** Escribir un nuevo script acorde a las especificaciones incluidas en el apartado 1.3.1.2 de esta SRS. Posteriormente, el usuario podrá guardar este script en disco.
- **Entrada:** El usuario escribirá un script por teclado sobre una interfaz gráfica diseñada para el caso. El usuario podrá guardar el script mediante click de ratón, insertando por teclado el nombre del fichero de texto destino. No debe haber otro script con el mismo nombre en el directorio seleccionado.
- **Salida:** Se generará un fichero de texto cuyo contenido será el script escrito por el usuario. Este fichero de texto se almacenará en el disco del usuario.
- **Origen:** Usuario del sistema.
- **Destino:** Sistema.
- **Necesita:** Los scripts harán referencia a catálogos dentro del sistema, por lo que será necesario mostrar al usuario por pantalla en todo momento un listado de catálogos.
- **Acción:** El usuario escribirá un script acorde a las especificaciones del apartado 1.3.1.2. El sistema copiará el script escrito por el usuario en un fichero de texto y lo guardará en disco con el nombre seleccionado.

- **Precondición:** No se requiere ninguna precondición en el estado del sistema.
- **Postcondición:** No se produce ningún cambio en el estado del sistema.
- **Efectos laterales:** -

c) Cargar script

- **Función:** Cargar script.
- **Prioridad:** Alta.
- **Estabilidad:** Alta.
- **Descripción:** Permite cargar desde disco un script previamente escrito.
- **Entrada:** Petición de este caso de uso mediante click de ratón. El usuario deberá elegir mediante ratón el fichero de texto que quiere cargar.
- **Salida:** Se mostrará por pantalla el script guardado previamente en el fichero de texto.
- **Origen:** Usuario del sistema.
- **Destino:** Sistema.
- **Necesita:** Un fichero de texto con un script.
- **Acción:** El sistema copiará el script escrito por el usuario desde un fichero de texto y lo mostrará por pantalla. El resultado debe ser el mismo que escribir el script, es decir, el usuario hará ejecutar el script recién cargado.
- **Precondición:** No se requiere ninguna precondición en el estado del sistema.
- **Postcondición:** No se produce ningún cambio en el estado del sistema.
- **Efectos laterales:** -

d) Escribir y guardar función

- **Función:** Escribir y guardar función.
- **Prioridad:** Media.
- **Estabilidad:** Alta.

- **Descripción:** Escribir una nueva función acorde a las especificaciones incluidas en el apartado 1.3.1.2 de esta SRS. Posteriormente, el usuario podrá guardar esta función en disco.
- **Entrada:** El usuario escribirá una función por teclado sobre una interfaz gráfica diseñada para el caso. El usuario podrá guardar la función mediante click de ratón. No debe existir otra función con el mismo nombre en el directorio seleccionado.
- **Salida:** Se generará un fichero de texto "nombre_de_tu_función.f" cuyo contenido será la función escrita por el usuario. Este fichero de texto se almacenará en el disco del usuario.
- **Origen:** Usuario del sistema.
- **Destino:** Sistema.
- **Necesita:** Este caso de uso no necesita no requiere ningún otro interfaz.
- **Acción:** El usuario escribirá una función acorde a las especificaciones del apartado 1.3.1.2. El sistema copiará la función escrita por el usuario en un fichero de texto y la guardará en disco con el nombre de la función y la extensión ".f".
- **Precondición:** No se requiere ninguna precondición en el estado del sistema.
- **Postcondición:** No se produce ningún cambio en el estado del sistema.
- **Efectos laterales:** -

e) Cargar función

- **Función:** Cargar función.
- **Prioridad:** Alta.
- **Estabilidad:** Alta.
- **Descripción:** Permite cargar desde disco una función previamente escrita para mostrarla por pantalla. No confundir con hacer uso de una función en un script.
- **Entrada:** Petición de este caso de uso mediante click de ratón. El usuario deberá elegir mediante ratón el fichero de texto que quiere cargar.

- **Salida:** Se mostrará por pantalla la función guardada previamente en el fichero de texto.
- **Origen:** Usuario del sistema.
- **Destino:** Sistema.
- **Necesita:** Un fichero de texto con una función.
- **Acción:** El sistema copiará la función escrita por el usuario desde un fichero de texto y la mostrará por pantalla.
- **Precondición:** No se requiere ninguna precondición en el estado del sistema.
- **Postcondición:** No se produce ningún cambio en el estado del sistema.
- **Efectos laterales:** -

1.3.3 Requisitos de rendimiento

La aplicación deberá trabajar con gigantescos catálogos de estrellas que pueden llegar a ocupar varios gigas de espacio en disco. Tanto para procesarlos (importarlos o exportarlos), como para hacer minería de datos sobre ellos, se requerirán tiempos de ejecución razonables, por lo que será necesario el uso de técnicas de programación y algoritmia para conseguir toda la eficiencia posible y ajustar al máximo dichos tiempos de ejecución. Uno de los principales objetivos de esta aplicación es precisamente este, acelerar de forma notable ciertos procedimientos que en la actualidad pueden llegar a tardar semanas y no se pueden realizar de forma automatizada. Aun así, es imposible actualmente establecer metas demasiado concretas al respecto dado la inexistencia de precedentes en esta tarea, la diversidad en el tamaño y en el contenido de los catálogos y la minería de datos que sobre ellos se quiera realizar (por ejemplo, catálogos del mismo tamaño pueden tener localizadas sus estrellas en hemisferios distintos, por lo que a la hora de cruzarlos usando como referencia la posición de estas estrellas, el tiempo de ejecución es mínimo al no poderse crear ninguna pareja válida).

En el mismo sentido, la aplicación no deberá quedarse bloqueada al ejecutar alguna de estas tareas, por lo que será necesario utilizar técnicas de programación multihilo en los casos de uso que requieran una mayor carga del sistema.

1.3.4 Requisitos del banco de datos

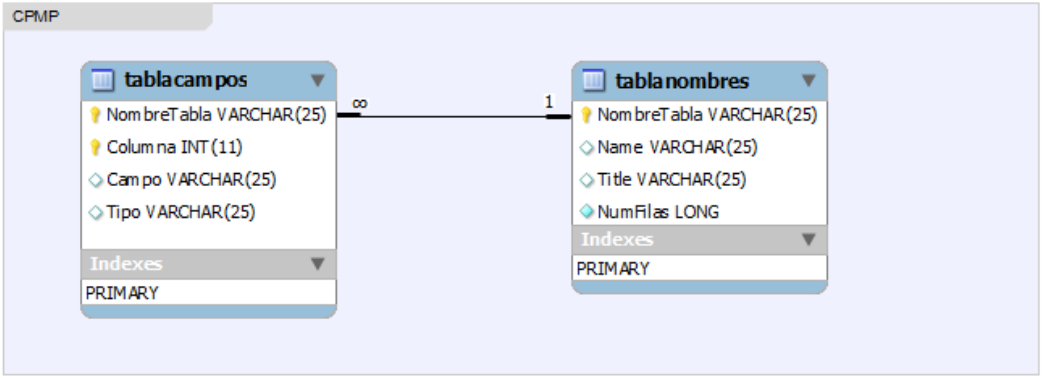


Figura 1 Diagrama Base de Datos

Estas dos tablas representan los cimientos sobre los que está construida toda la aplicación. Por un lado, tenemos "tablanombres". Esta tabla contiene el nombre de cada catálogo, su nomenclatura dentro de VizieR, y la que usa internamente en CPMP (nombre elegido por el usuario). Cada uno de estos catálogos tiene asociados una serie de entradas en "tablacampo" (se usan claves extranjeras y borrado en cascada). Cada una de estas entradas representa un campo de las estrellas que están incluidas en cada catálogo. De esta forma podremos manejar cualquier tipo de catálogo, ya que extraeremos de cierta manera los campos de las estrellas, sean cuales sean estos, de forma dinámica cuando se está realizando el parseo del catálogo.

1.3.5 Restricciones de diseño

El proyecto se codificará en el lenguaje de programación Java con soporte inicial para la persistencia de datos en MySQL, que podrá ser expandido en un futuro con otros motores de persistencia. El diseño de la aplicación será

representado con UML y el Plan de Proyecto Software se realizará siguiendo los estándares ISO del IEEE correspondientes.

Esta aplicación, debido a sus características y a su finalidad académica, no entrará en conflicto con ninguna normativa legal que restrinja su diseño, u obligue a realizar algún tipo de registro de uso para su posterior auditoria.

1.3.6 Atributos del sistema software

Los atributos principales del sistema son la velocidad y estabilidad de las funciones que lo componen. Además, nuestro sistema va a ser diseñado de tal forma que se favorezca la mantenibilidad y la evolución de este.

Las razones por las cuales el sistema debe ser veloz ya han sido explicadas con anterioridad en este documento, por lo que nos centraremos dentro de este apartado en los restantes atributos.

Estabilidad: El sistema debe ser estable, dado que va a realizar tareas que pueden durar grandes lapsos de tiempo. Un sistema inestable sería completamente inútil dados nuestros objetivos. Si uno de las principales metas es acelerar de forma notable la minería de datos sobre catálogos, supondría un enorme problema tener que volver a repetir ciertas operaciones muy costosas sobre los catálogos debido a que se produzcan errores durante estas.

Mantenibilidad: El sistema software deberá ser mantenible y fácilmente evolucionable, ya que dado su carácter académico, sería muy interesante que los profesionales de la astronomía que conforman la audiencia de la aplicación, puedan extender de la forma más sencilla posible el alcance funcional de este sistema. Además, en un futuro, este mismo sistema informático podría ser incluido en un sistema software mayor o servir de base para un nuevo proyecto de Sistemas Informáticos.

Portabilidad: El sistema deberá poder ser ejecutado en el mayor número de máquinas posibles, y no está destinado a un sistema operativo concreto. Para ello, se utilizará el lenguaje de programación Java dado la utilidad a tal propósito de su máquina virtual. Se considera como posible requisito futuro la posibilidad de poder configurar la aplicación en distintos idiomas.

Fiabilidad: El sistema debe ser completamente fiable puesto que será desarrollado para la asignatura de Sistemas Informáticos, dentro del plan de estudios de Ingeniería informática de la Universidad Complutense de Madrid. Un sistema que no funcione o lo haga de forma incorrecta supondría no aprobar dicha asignatura.

Otros aspectos como la seguridad del sistema no han sido considerados importantes dado las características de esta aplicación.

CAPÍTULO 2
PLAN DE PROYECTO SOFTWARE

2.1 INTRODUCCIÓN

2.1.1 Propósito

El uso de técnicas de Ingeniería del Software en el desarrollo de una nueva aplicación nos proporciona un enfoque sistemático, disciplinado y cuantificable tanto del proceso de desarrollo en sí, como del manejo y mantenimiento de dicha aplicación. Estas técnicas nos permitirán garantizar el desarrollo de un software de calidad en un espacio de tiempo predefinido y con unos costes razonables.

El presente plan de proyecto pretende recoger toda la información referente a la gestión del proyecto, así como servir de guía para todo el proceso de desarrollo.

2.1.2 Ámbito del proyecto y objetivos

En esta sección se ofrece un breve resumen del ámbito y perspectiva del proyecto. Para obtener una información más detallada se podrá consultar la especificación de requisitos software que acompaña a este documento.

2.1.2.1 Declaración del ámbito

Nuestro sistema informático, al que de ahora en adelante nos referiremos como CPMP (Common Proper Motion Pairs) estará compuesto por dos módulos principales. El primero se tratará de un sistema de gestión de catálogos de estrellas. Este modulo permitirá al usuario importar masivamente, a partir de ficheros de texto, conjuntos de estrellas. Dichos conjuntos de estrellas estarán agrupados dentro de la aplicación en catálogos según las especificaciones de VizieR (servicio de catálogos astronómicos proporcionado por el Centre de Données Astronomiques de Strasbourg (CDS)) y deberán persistir en el equipo del usuario.

A la hora de trabajar con conjuntos de estrellas nos referiremos siempre a estos como catálogos.

Una vez importado uno o varios catálogos, el usuario podrá gestionarlos por separado o conjuntamente, teniendo disponible por pantalla un listado de estos. Individualmente, tendrá la opción de renombrar un catálogo, borrarlo, exportarlo de nuevo a un fichero de texto, mostrar sus campos o mostrarlo por pantalla. Si el usuario ha elegido esta última opción, la aplicación le dará la posibilidad de mostrar una imagen de cada una de las estrellas incluidas en dicho catálogo, siempre y cuando hayan sido fotografiadas previamente y estén incluidas dentro de Aladin, otro de los servicios web del Centre de Données Astronomiques de Strasbourg.

Si el usuario desea realizar operaciones sobre más de un catálogo a la vez, tendrá la opción de unir dos catálogos siempre que sus estrellas estén igualmente representadas dentro de la aplicación, o cruzarlos para encontrar pares de estrellas que cumplan unas determinadas características de proximidad.

El segundo modulo consistirá en un entorno de trabajo, una IDE, para el desarrollo de scripts. Estos scripts constituirán una manera potente, flexible y veloz de trabajar con los catálogos y realizar minería de datos sobre ellos. El usuario dispondrá en principio de 10 operaciones básicas, pero podrá construir a partir de ellas funciones más complejas que usar en los scripts. Estas funciones podrán ser guardadas y utilizadas a posteriori.

CPMP proporcionará al usuario una herramienta simple, intuitiva, automatizada y ante todo, extremadamente veloz, para gestionar cualquier tipo de catálogos de estrellas de forma unificada.

2.1.2.2 Funciones principales

Las principales funciones de nuestro sistema, agrupadas en los módulos anteriormente descritos son:

Módulo de gestión de catálogos:

- Importar catálogo VizieR.
- Crear cabecera VizieR.

- Exportar catálogo.
- Listar catálogos.
- Mostrar campos de un catálogo.
- Mostrar catálogo.
- Mostrar estrella.
- Renombrar catálogo.
- Borrar catálogo.
- Unir dos catálogos.
- Cruzar dos catálogos.
- Resetear aplicación.
- Dividir catálogos.

Módulo de desarrollo de scripts:

- Escribir y ejecutar script.
- Escribir y guardar script.
- Cargar script.
- Escribir y guardar función.
- Cargar función.

2.1.2.3 Aspectos de rendimiento

La aplicación deberá trabajar con gigantescos catálogos de estrellas que pueden llegar a ocupar varios gigas de espacio en disco. Tanto para procesarlos (importarlos o exportarlos), como para hacer minería de datos sobre ellos, se requerirán tiempos de ejecución razonables, por lo que será necesario el uso de técnicas de programación y algoritmia para conseguir toda la eficiencia posible y ajustar al máximo dichos tiempos de ejecución. Uno de los principales objetivos de esta aplicación es precisamente este, acelerar de forma notable ciertos procedimientos que en la actualidad pueden llegar a tardar semanas y no se pueden realizar de forma automatizada. Aun así, es imposible actualmente establecer metas demasiado concretas al respecto dado la inexistencia de

precedentes en esta tarea, la diversidad en el tamaño y en el contenido de los catálogos y la minería de datos que sobre ellos se quiera realizar (por ejemplo, catálogos del mismo tamaños pueden tener localizadas sus estrellas en hemisferios distintos, por lo que a la hora de cruzarlos usando como referencia la posición de estas estrellas, el tiempo de ejecución es mínimo al no poderse crear ninguna pareja válida). En el mismo sentido, la aplicación no deberá quedarse bloqueada al ejecutar alguna de estas tareas, por lo que será necesario utilizar técnicas de programación multihilo en los casos de uso que requieran una mayor carga del sistema.

2.1.2.4 Restricciones y técnicas de gestión

Las restricciones impuestas a nuestro proyecto software son las descritas en la normativa de la asignatura Sistemas Informáticos de la Facultad de Informática de la Universidad Complutense de Madrid (Código UCM-360-98-434).Dicha normativa puede encontrarse en el siguiente enlace:

http://www.fdi.ucm.es/SI/Normativa-SI-Junta05Nov07.pdf

Aparte de dicha normativa general, para el curso académico 2009-2010 se han establecido las siguientes restricciones específicas para las fechas de entrega:

- Entrega del borrador del proyecto al director del mismo: Viernes 11 de junio de 2010.
- Entrega de la memoria final en secretaría de alumnos: Viernes 2 de julio de 2010.
- Exposiciones públicas de los trabajos que no optan a MH: Del 5 al 8 de julio de 2010.
- Exposiciones públicas al tribunal de MH: Del 8 al 9 de julio de 2010.

2.1.3 Modelo de proceso

Los modelos de proceso definen un conjunto de actividades, acciones, tareas, fundamentos y productos de trabajo que se requieren para desarrollar software de alta calidad. Proporcionan estabilidad, control, y organización a una actividad que, si no se controla, puede volverse caótica.

La elección de un modelo de proceso adecuado para el proyecto a desarrollar es una de las partes más importantes de la planificación del proyecto.

Creemos que el modelo a escoger debe ser evolutivo, ya que partimos de un conjunto esencial de requisitos que deberá ser definido en detalle a la vez que se produce el desarrollo del proyecto, es decir, iremos elaborando versiones cada vez más completas del software.

Dentro de los modelos de proceso evolutivos nos hemos decantado por un modelo en Espiral de Boehm ya que, a diferencia del Proceso Unificado de Desarrollo, no está tan ligado al método e incluye explícitamente actividades de gestión de riesgos, aspecto crucial dada nuestra inexperiencia en desarrollo de esta clase de proyectos.

Nuestro modelo de proceso se descompondrá en seis actividades estructurales adaptando la definición de Pressman [P05]:

1. Comunicación con el director del proyecto:

Se establecerá la comunicación entre los desarrolladores y el director del proyecto.

2. Planificación:

Se definirán los recursos, la estimación de tiempo y la división de tareas entre los miembros del grupo.

3. Análisis de riesgos:

Se evaluarán los riesgos técnicos y de gestión que puedan afectar al proyecto.

4. Ingeniería:

Esta fase abordará las tareas requeridas para la construcción de una o más representaciones de nuestra aplicación.

5. Construcción y adaptación:

Desarrollo, pruebas e instalación del proyecto.

6. Evaluación:

El director del proyecto evaluará el trabajo realizado.

2.1.4 Referencias

Consultar la bibliografía al final del documento.

2.2 ESTIMACIONES DEL PROYECTO

2.2.1 Datos históricos

No se disponen de datos históricos para poder estimar el esfuerzo, el coste y la duración de nuestro proyecto.

2.2.2 Técnicas de estimación

Entre todas las técnicas de estimación hemos optado por la de "Descomposición del trabajo". Puesto que no disponemos de datos históricos del esfuerzo medio necesario para desarrollar un número determinado de líneas de código o un punto de función, consideramos que lo más lógico será usar una técnica de descomposición basada en nuestro modelo de proceso.

2.2.3 Estimaciones de esfuerzo, coste y duración

Estimación del esfuerzo

El esfuerzo total aparece en personas/día. La estimación del esfuerzo para nuestro proyecto puede verse en el apéndice 5 de este documento.

Estimación del coste

Dada la finalidad académica de nuestro proyecto software, las estimaciones de coste no son relevantes en nuestro proyecto.

Estimación de duración

Aunque la estimación de duración inicial está definida según el marco de la asignatura de Sistemas Informáticos, acordamos establecer, junto con el director del proyecto, un calendario más específico que en ningún caso entrará en conflicto con las restricciones temporales incluidas en el marco de la asignatura. Esta

estimación puede verse en los documentos incluidos en el apéndice 5 de este documento.

2.3 ESTRATEGIAS DE GESTIÓN DE RIESGO

Llamamos riesgo a todo aquello que pueda afectar negativamente al desarrollo de nuestro proyecto software. Aunque esta es una definición poco precisa, nos servirá para establecer un marco de gestión de riesgos adaptado dicho desarrollo.

Más concretamente este apartado del plan de proyecto se encargará de valorar que riesgos merecen más atención, ya sea por su probabilidad de aparición o por el impacto que puedan tener sobre el proyecto, con el objetivo de gestionarlos adecuadamente. Podemos afrontar la gestión del riesgo con dos tipos de estrategias:

a) Estrategia reactiva: Se analiza el proyecto buscando posibles riesgos, asignando a cada uno una serie recursos de manera que, si alguna vez se presentan, se puedan sofocar. Si estos mecanismos paliativos fracasaran, entraría en acción la gestión de crisis.

b) Estrategia proactiva: Comienza antes que los trabajos técnicos. Su labor es prevenir el daño evaluando probabilidad, impacto y prioridad de los riesgos potenciales para, a continuación, realizar un plan de gestión de riesgo.

Para nuestro proyecto, teniendo en cuenta factores como la falta de experiencia y recursos debido al carácter académico del mismo, consideramos que una estrategia proactiva es más adecuada para evitar la incidencia de los riesgos.

Podemos clasificar los riesgos como conocidos, desconocidos e impredecibles. Una de nuestras principales labores será recortar el número de riesgos desconocidos mediante una exhaustiva identificación.

La metodología que utilizaremos para la gestión de riesgo es la contenida en el documento *"Software Risk Management: Principles and Practices"* [B91] ya que, sin ser la aproximación más completa, estimamos que cumple de sobra los objetivos marcados.

2.3.1 Valoración del riesgo

La valoración del riesgo según Boehm [B91] está compuesta por tres fases:

- ◊ Identificación del riesgo.
- ◊ Análisis del riesgo.
- ◊ Priorización del riesgo.

2.3.1.1 Identificación del riesgo

La identificación del riesgo produce listas de elementos de riesgo específicas para el proyecto, los cuales comprometen seriamente su éxito. Las técnicas de identificación del riesgo son:

- ◊ **Uso de listas de comprobación de elementos de riesgo**.
- ◊ **Análisis y la comparación.**

Existen diversas listas de comprobación, siendo una de las más útiles el Top 10 Software Risk Ítems de Boehm [B91]. Sin embargo, consideramos que los contenidos de esta clase de listas son excesivos o no concuerdan con el tipo de proyecto que nosotros vamos a desarrollar, por lo que, aun teniéndolas como obligada referencia, nos decantaremos por un análisis un poco más realista y ajustado a nuestra circunstancia de estudiantes y a las indicaciones del director del proyecto.

Basándonos en nuestra heurística, estos son los riesgos identificados para el proyecto:

1. Descomposición del grupo

Este proyecto está pensado para que sea realizado por un número exacto de 3 personas. El fallo de alguno de los miembros puede ser desastroso para la consecución final del mismo.

2. Posibilidad de bloqueo

A la hora de desarrollar el proyecto es necesario realizar labores de investigación que conduzcan a alcanzar los diversos objetivos marcados. Una

situación de bloqueo o un resultado insatisfactorio en dicha investigación podrán retrasar o imposibilitar la consecución de dichos objetivos.

3. Planificación poco realista o poco equilibrada

Nuestra poca o nula experiencia en el desarrollo de este tipo de sistemas hace que podamos tener problemas al estimar el esfuerzo necesario y por ende, la planificación o realizar una división de la carga de trabajo poco equilibrada.

4. Poca experiencia con las herramientas

Para la realización del proyecto manejaremos herramientas como los programas Eclipse, Tortoise, servidores SVN, MySQL, VizieR, Aladin,..., algunos de ellos novedosos en nuestra formación. Una mala asimilación de su entorno y funcionalidad puede entorpecer sobremanera los avances y desajustar toda la planificación.

5. Desajuste en la planificación

Es bastante probable que, por algún motivo impredecible, nos sea imposible cumplir uno o más de los plazos marcados, lo que afectará al resto de la planificación.

6. Problema con los laboratorios

Algunas de las herramientas que vamos a utilizar son de pago, por lo que solo podremos acceder a ellas mediante los equipos de la facultad. Además, en general, el acceso al software que necesitaremos para el desarrollo de la aplicación está restringido a ciertos laboratorios, por lo que cualquier problema con el acceso a dichos equipos o con el software instalado en ellos será un riesgo para nuestro proyecto.

7. Problemas de accesibilidad al director de proyecto

El director de proyecto tiene unos horarios limitados de atención a los alumnos. Se puede producir la circunstancia de que, debido a nuestros horarios, nos sea difícil establecer reuniones con él, o bien que debido a algún tipo de percance, el director de proyecto no esté disponible durante una larga temporada.

8. Problemas para combinar horarios

Los integrantes del grupo tenemos horarios y asignaturas distintas en ambos cuatrimestres. Conseguir solapar horarios de tal manera que podamos

reunirnos todos los miembros del grupo el tiempo suficiente es un problema que tendremos que afrontar.

9. Problemas por exceso de carga lectiva

Los integrantes del grupo deberán compaginar el desarrollo del proyecto con su respectiva carga lectiva. Un exceso de la misma supondrá una menor dedicación al proyecto o la imposibilidad de cumplir los plazos establecidas.

10. Caída del repositorio donde se almacenan las versiones

No se puede evitar la caída del servidor de versiones ya que está contratado a una empresa externa. Esto supondrá la imposibilidad de acceder a dichas versiones para continuar con el trabajo.

2.3.1.2 Análisis del riesgo

El análisis de riesgos determina la probabilidad e impacto asignados a cada riesgo. La probabilidad indica las posibilidades de que se haga real y el impacto, la gravedad de sus consecuencias. La asignación de probabilidades e impacto será estimada directamente, utilizando como referencia y ponderación la tabla del SQAS-SEI[5], una ampliación del trabajo de Boehm:

Probabilidad	Descripción	Impacto	Consecuencia
Frecuente	No sucede por sorpresa, ocurrirá muchas veces (Frecuencia por año > 1)	Catastrófico	Más de 6 meses de desplazamiento en la programación, más del 10% de sobrecoste, más del 10% de reducción en la productividad
Probable	Ocurre repetidamente, será un evento esperado (Frecuencia por año > 1 -10 $^{-1}$)	Crítico	Menos de 6 meses de desplazamiento en la programación, menos del 10% de

			sobrecoste, menos del 10% de reducción en la productividad
Ocasional	Podría ocurrir alguna vez (Frecuencia por año > 10^{-1} -10^{-2})	Serio	Menos de 3 meses de desplazamiento en la programación, menos del 5% de sobrecoste, menos del 5% de reducción en la productividad
Remoto	Improbable pero plausible (Frecuencia por año > 10^{-2} -10^{-4})	Menor	Menos de 1 meses de desplazamiento en la programación, menos del 2% de sobrecoste, menos del 2% de reducción en la productividad
Improbable	Prácticamente imposible (Frecuencia por año > 10^{-4} -10^{-5})	Sin Importancia	Impacto insignificante

Elemento de riesgo	Probabilidad	Impacto
Descomposición del grupo	Ocasional	Catastrófico
Posibilidad de bloqueo	Probable	Crítico

Planificación poco realista o poco equilibrada	Frecuente	Serio
Poca experiencia con las herramientas	Frecuente	Menor
Desajuste en la planificación	Probable	Serio
Problema con los laboratorios	Frecuente	Menor
Problemas de accesibilidad al director del proyecto	Improbable	Crítico
Problemas para combinar horarios	Probable	Menor
Problemas por exceso de carga lectiva	Ocasional	Crítico
Caída del repositorio donde se almacenan las versiones del proyecto	Remota	Sin Importancia

2.3.1.3 Priorización del riesgo

La priorización de riesgos produce una lista ordenada de elementos de riesgo identificados y analizados. Para "graduar" el riesgo, recurriremos una vez más a una tabla del SQAS-SEI [SQAS22.01.00-2002]:

	Frecuente	Probable	Ocasional	Remota	Improbable
Catastrófico	IN	IN	A	A	M
Crítico	IN	IN	A	M	B

Serio	A	A	M	B	T
Menor	M	M	B	T	T
Sin Importancia	M	B	T	T	T

Leyenda:

IN: Intolerable

A: Alto

M: Medio

B: Bajo

T: Tolerable

Atendiendo a esta tabla clasificatoria, nuestra priorización quedaría:

Elementos de riesgo	Nivel de riesgo
Posibilidad de bloqueo	Intolerable
Descomposición del grupo	Alto
Planificación poco realista o poco equilibrada	Alto
Desajuste en la planificación	Alto
Problemas por exceso de carga lectiva	Alto
Problemas con los laboratorios	Medio
Problemas para combinar horarios	Medio
Poca experiencia con la herramienta	Medio

Problemas de accesibilidad al director de proyecto	Bajo
Caída del repositorio donde se almacenan las versiones del proyecto	Tolerable

2.3.2 Plan de gestión del riesgo

El plan de gestión de riesgos convierte la información sobre riesgos en decisiones y acciones para el presente y el futuro. Esta información la codificaremos en un Plan de Reducción, Supervisión y Gestión del Riesgo (RSGR). Para no hacer la gestión de riesgos un proceso inviable, aplicaremos el *Principio de Pareto: el 80% del riesgo real se debe al 20% de los riesgos identificados*, por lo que solo gestionaremos los riesgos más importantes.

Posibilidad de bloqueo

- **Reducción :**

  Se intentará trabajar con el mayor número de referencias bibliográficas posibles, de tal forma que esté al alcance del grupo de desarrollo la máxima información posible sobre la materia a investigar. Se intentará siempre plantear caminos alternativos a la hora de plantear la solución a un problema.

- **Supervisión :**

  Dentro de las reuniones periódicas con el director del proyecto se expondrán los avances conseguidos y se analizara con detenimiento hacia dónde conduce la línea de trabajo.

- **Gestión :**

  En caso de que el bloqueo sea absoluto, y no existan rutas alternativas, expondremos en la memoria de forma detallada los problemas encontrados, las soluciones buscadas, y el resultado insatisfactorio de nuestro trabajo.

Descomposición del grupo

◊ **Reducción :**

Se intentará que ninguno de los componentes del grupo tenga otras obligaciones que no sean las lectivas. Se intentará a su vez que los integrantes del grupo se conozcan y estén acostumbrados a trabajar en equipo. Se intentará que las personas seleccionadas sean diligentes y capaces de solucionar problemas de forma individual.

◊ **Supervisión:**

Se realizarán reuniones periódicas en las que los miembros del grupo expondrán su opinión sobre el desarrollo del proyecto y la carga asignada a cada uno.

◊ **Gestión :**

Si el riesgo se convierte en una realidad y uno de los miembros no cumple con los objetivos marcados por el grupo, el resto de miembros tendrá la obligación de hablar con él de forma conjunta para incentivarle de la manera que estimen oportuna. Si se diera el caso de que uno de los miembros abandonara el proyecto, se hablaría con el director del mismo para intentar reducir el alcance funcional de la aplicación en la medida de lo posible, de modo que fuera posible realizar el proyecto en los plazos establecidos.

Planificación poco realista o poco equilibrada

◊ **Reducción :**

Se intentará que, al menos, uno de los miembros del grupo tenga experiencia planificando proyectos. Aun así, se hará un exhaustivo análisis del proyecto y del tiempo disponible, dejando un amplio margen de tiempo para la realización de cada tarea del proceso, en pos de evitar que una estimación demasiado ajustada del esfuerzo necesario lleve al traste toda la planificación.

◊ **Supervisión:**

Se consultará las veces que sean necesarias al profesor y, utilizando como modelo uno de los módulos, se comprobará si el resto de la planificación es realista.

- **Gestión :**

  Si el riesgo se convierte en una realidad, aumentaremos el esfuerzo en forma de personal en todas las tareas hasta donde sea pertinente.

2.4 PLANIFICACIÓN

El modelo de proceso elegido para desarrollar nuestra planificación temporal es la espiral de Boehm [B91], en la que diferenciamos las siguientes actividades estructurales que compondrán nuestra estructura de descomposición del trabajo (EDT):

- **Comunicación con el cliente:** Reuniones con el director del proyecto para comentar el avance en el desarrollo de la aplicación.
- **Plan y Gestión:** Engloba Planificación Temporal y Gestión de los Riesgos.
- **Ingeniería:** Este apartado engloba los siguientes procesos:
  - **Análisis del Proyecto:** analizar el dominio del sistema y obtener una descripción detallada del alcance funcional y los requisitos de la aplicación.
  - **Diseño:** obtener una representación de cómo va a estar implementada la aplicación.
- **Construcción y Adaptación:** Tareas dedicadas a la generación de código de los diferentes módulos de la aplicación (codificación), su prueba y el ensamblado de estos módulos.
  - **Codificación**
  - **Prueba**
  - **Ensamblaje**
- **Evaluación:** Revisión por parte del director del proyecto.

2.4.1 Estructura de la Descomposición del trabajo

La estructura de descomposición de trabajo (EDT) representa gráficamente una matriz donde aparecen las actividades estructurales comentadas y las funciones o módulos en los que se divide nuestro proyecto.

Los grupos de trabajo sobre los módulos son variables en función de los conocimientos requeridos para desarrollarlos, además, son dependientes de las cargas de trabajo.

Los módulos que se iteran en la EDT son *Gestión de catálogos* (Gc) y *Desarrollo de scripts* (Ds). En el caso del primero, dada su complejidad, lo dividiremos, a su vez, en tres unidades funcionales:

- Núcleo funcional de la gestión de catálogos.
- Desarrollo de las funcionalidades asociadas a la minería de datos.
- Implementación de la GUI asociada a este módulo y desarrollo del menú de ayuda al usuario.

Además, a la hora de mostrar la EDT, añadiremos un módulo al que llamaremos Proyecto que reflejará las actividades relacionadas con ámbitos más generales del mismo.

Nuestra EDT puede consultarse en el Apéndice 4 de este documento.

2.4.2 Grafico Gantt

El Gráfico de Gantt es un gráfico de tiempo que muestra la evolución de las tareas que componen nuestro proyecto. En él se detallan la fecha de inicio, la duración y el esfuerzo de cada tarea, así como la precedencia existente entre ellas y los recursos asignados a cada una.

Todas las tareas del proyecto se listan en la columna de la izquierda. Las barras horizontales indican la duración de cada tarea. Cuando aparecen múltiples barras al mismo tiempo en la planificación temporal, implican concurrencia de tareas. Las estrellas indican hitos.

Nuestro Gráfico de Gantt se puede ver en el Apéndice 6.

2.4.3 Tabla de uso de recursos

Incluido en el Apéndice 7.

2.5 RECURSOS

2.5.1 Personal

Los recursos humanos disponibles para la realización del proyecto serán los integrantes del grupo. A continuación se muestra un listado con el nombre y la disponibilidad del recurso.

- Blanca Collado Iglesias.

  De lunes a viernes: 16:00-20:00. Lunes 11:00-12:00.

- Antonio Fernández Sánchez.

  De lunes a viernes: 12:00-17:00. Lunes 11:00-12:00.

- Sara Pozuelo González.

  De lunes a viernes: 15:00-19:00. Lunes 11:00-12:00.

2.5.2 Hardware y Software

En este apartado se describen los recursos hardware y software asignados al proyecto, así como su disponibilidad.

Recursos hardware

- **Ordenadores de la facultad de informática:**

  Descripción del recurso: Pentium IV a 1,7GHz con 256MB de RAM.

  Disponibilidad: La disponibilidad de estos recursos no es total, ya que depende de los horarios en los que los laboratorios están disponibles para la realización de prácticas libre. Además estos recursos también son usados por otros compañeros de la facultad, por lo que no se asegura la disponibilidad del recurso en los horarios referenciados anteriormente. Asimismo, dadas las necesidades software de nuestra aplicación, sólo se podrá utilizar un subconjunto de ellos. Para más información, ver Tabla de Horarios de Laboratorio.

- **Ordenadores personales de cada miembro del grupo:**

Descripción del recurso:

Antonio: Pentium Intel Centrino a 1.7 GHz, 1512 Mb RAM, DD 60 Gb

Blanca: Pentium Intel Core2 Duo t7200, 1000 Mb RAM, DD 160 Gb

Sara: Pentium Intel Core2 Duo p8600 2.4 Ghz, 4Gb RAM, DD 300 Gb

Disponibilidad: Total.

Recursos software

- **Eclipse**

  Descripción del recurso: Eclipse es una plataforma Open Source independiente para el desarrollo software. Esta plataforma se usa habitualmente para desarrollar aplicaciones J2EE. Destaca por su gran número de plugins disponibles.

  Disponibilidad: Disponibilidad absoluta, ya que cada miembro del grupo dispone de dicho software en su ordenador personal, y éste es gratuito. Además éste recurso también está disponible en los laboratorios de la universidad.

- **Subclipse**

  Descripción del recurso: Es un *plugin* para Eclipse que permite interactuar con un servidor SVN. Esto nos permite llevar un control de versiones del proyecto y una gestión individualizada de cada componente.

  Disponibilidad: Total. Es descargable gratuitamente por Internet. Se instala fácilmente desde el propio eclipse.

- **Servidor Google Code para Subversion**

  Descripción del recurso: Subversion es un software de sistema de control de versiones diseñado específicamente para reemplazar al popular CVS, el cual posee varias deficiencias. Es software libre bajo una licencia de tipo Apache/BSD y se le conoce también como SVN por ser ese el nombre de la herramienta de línea de comandos. Una característica importante de Subversion es que, a diferencia de CVS, los archivos versionados no tienen cada uno un número de revisión independiente. En cambio, todo el repositorio tiene un único número de versión que identifica un estado común de todos los archivos del repositorio en cierto punto del tiempo. Este servidor nos permitirá gestionar, documentar y programar nuestro

proyecto con gran facilidad. En nuestro proyecto usaremos como servidor de Subversion Google Code.

Disponibilidad: Total, aunque ajeno al control del grupo.

◊ **Servidor de Base de datos MySQL**

Descripción del recurso: MySQL es un sistema de gestión de base de datos relacional, multihilo y multiusuario.

Disponibilidad: Total.

◊ **Microsoft Project**

Descripción del recurso: Microsoft Project (o MSP) es un software de administración de proyectos diseñado, desarrollado y comercializado por Microsoft para asistir a administradores de proyectos en el desarrollo de planes, asignación de recursos a tareas, dar seguimiento al progreso, administrar presupuesto y analizar cargas de trabajo.

Disponibilidad: Este recurso está presente en los ordenadores y laboratorios de la facultad, por ello su disponibilidad está restringida por la disponibilidad de los mismos. Estará disponible durante la apertura de los laboratorios para la realización de prácticas libres. También está disponible en Internet a través de MSDN Academy.

◊ **Editores de texto**

Descripción del recurso: herramientas con las que serán realizados los informes y memorias del proyecto.

Disponibilidad: Disponibilidad absoluta, ya que cada miembro del grupo dispone de dicho software en su ordenador personal. Además éste recurso está presente en los ordenadores de la facultad.

◊ **Rational Rose**

Descripción del recurso: Herramienta CASE para la creación de diagramas UML.

Disponibilidad: Este recurso está presente en los ordenadores y laboratorios de la facultad, por ello su disponibilidad está restringida por la disponibilidad de los mismos.

- **Software Ideas Modeler**

  Descripción del recurso: Herramienta CASE para la creación de diagramas UML.

  Disponibilidad: Este recurso tiene licencia gratuita y está disponible en la web www.softwareideas.net/ .

2.5.3 Lista de recursos

Personal

- Blanca Collado Iglesias.
- Antonio Fernández Sánchez.
- Sara Pozuelo González.

Hardware

- Ordenadores de la facultad: Pentium IV a 1,7GHz, 256MB de RAM.
- Pentium Intel Centrino a 1.7 GHz, 1512 Mb RAM, DD 60 Gb.
- Pentium Inter Core2 Duo t7200, 1000 Mb RAM, DD 160 Gb.
- Pentium Intel Core2 Duo p8600 2.4 Ghz, 4Gb RAM, DD 300 Gb.

Software

- Eclipse
- Sublicpse
- Servidor Google Code para Subversion.
- Servidor de Base de datos Mysql.
- Microsoft Project.
- Editores de texto.
- Rational Rose.
- Software Ideas Modeler.

2.6 ORGANIZACIÓN DEL PERSONAL

2.6.1 Estructura de equipo

En cualquier proyecto software a mediana o gran escala, el gestor del proyecto es la persona encargada de organizar el personal. En nuestro caso, este rol será ejercido por el profesor de la asignatura Sistemas Informáticos, que sin embargo, dejará la elección de dicha organización al propio grupo de desarrollo. A la hora de escoger una estructura de equipo hemos analizado, en función de las características de este proyecto, los factores que se identifican en [M81] para encontrar la gestión de grupo más apropiada:

Dificultad del problema: Analizando las características del proyecto llegamos a la conclusión de que su complejidad algorítmica es baja. Por ello no es necesaria la mejor capacidad de solución de problemas que posee la estructura democrática descentralizada. Con lo que cabría pensar que este factor debería hacernos pensar en una estructura centralizada controlada. Sin embargo la inexperiencia de los miembros del equipo en el desarrollo de proyectos aplicando técnica de Ingeniería del Software, si recubre al proyecto de una dificultad considerable.

Tamaño en líneas de código (LDC) o puntos de función (PF): Estimamos que el tamaño del código será medio-bajo, ergo descentralizado controlado y descentralizado democrático son las estructuras que mejor se adaptan a este factor.

Duración del equipo: Dado que la estructura democrática descentralizada aumenta la duración del proyecto y esto puede ser un riesgo al tener una fecha fija de entrega, este factor apunta a las estructuras descentralizado controlado y centralizado controlado.

Modularidad del problema: Aunque hemos logrado separar el problema en dos módulos existen ligeras dependencias entre ellos. Es necesaria cierta comunicación y, por tanto, una estructura descentralizada controlada o descentralizada democrática.

Calidad y fiabilidad del sistema a construir: No es un sistema cuya fiabilidad sea crítica, como el software encargado del control de una central nuclear o de un avión. Sin embargo estos factores serán tenidos en cuenta a la hora de evaluar este proyecto dentro del contexto de la asignatura. En principio gracias, al periodo de pruebas asignado (el código también será probado por personas que no lo han desarrollado) y a las técnicas de Ingeniería del Software aplicadas, tales como la gestión de la configuración software, el número de defectos debería ser pequeño teniendo en cuenta que la complejidad de la aplicación es baja. Por ello, aunque este factor mejoraría indudablemente con una estructura centralizada controlada, no consideramos que excluya a las otras dos posibles estructuras.

Fecha de entrega: Este factor es muy importante en este proyecto, debido a su carácter académico. Este factor apunta, por tanto, a una estructura centralizada controlada.

Comunicación requerida en el proyecto: Un factor extremadamente importante en nuestro caso. Todo el mundo deberá ser consciente del desarrollo del trabajo del resto del equipo en cada momento. Seguramente durante todo el proceso de desarrollo surgirán infinidad de dudas que debemos estar dispuestos a solucionar unos a otros, puesto que, en muchos casos, nos enfrentaremos a situaciones y problemas desconocidos por todos. Esto exige un altísimo grado de comunicación entre todos los miembros del equipo. Esto nos hace pensar en la estructura democrática descentralizada.

De entre todos los factores, este último provocó nuestro interés por la estructura democrática descentralizada. Además nos atraía la idea de que hubiera una comunicación horizontal entre los miembros del grupo. Por último, ningún miembro del equipo tiene una gran experiencia en este tipo de proyectos como para hacer de él un buen "jefe" o gestor de proyecto, es decir, como para tomar decisiones sobre el desarrollo que en principio debieran ser mejores que las de cualquier otra persona. La unión de de todas estas circunstancias nos llevó a optar definitivamente por una estructura Democrática Descentralizada.

2.6.2 Informes de gestión

◊ Antonio Fernández Sánchez

De lunes a viernes: 12:00-17:00. Lunes 11:00-12:00

Experiencia: Asignaturas de Programación Orientada a Objetos, laboratorios de Programación 2 y 3, asignatura de Ingeniería del Software, asignatura de Bases de Datos.

◊ Blanca Collado Iglesias

De lunes a viernes: 16:00-20:00. Lunes 11:00-12:00

Experiencia: Asignaturas de Programación Orientada a Objetos, laboratorios de Programación 2 y 3, asignatura de Ingeniería del Software. 3 meses trabajando en INDRA.

◊ Sara Pozuelo González

De lunes a viernes: 15:00-19:00. Lunes 11:00-12:00

Experiencia: Asignaturas de Programación Orientada a Objetos, laboratorios de Programación 2 y 3, asignatura de Ingeniería del Software.

2.7. MECANISMOS DE SEGUIMIENTO Y CONTROL

Los elementos que componen toda la información producida como parte del proceso de ingeniería del software se denominan colectivamente "configuración del software". Dado que la configuración software es la única representación tangible de un programa o sistema software, debe ser controlada para conservar su exactitud, mantener la información actualizada, y asegurar una información clara y concisa conforme avanzamos paso tras paso en el proceso de ingeniería del software.

2.7.1 Gestión de calidad

Las técnicas de ingeniería del software para conseguir un software de alta calidad se denominan Garantía de Calidad del Software (SQA: Software Quality

Assurance) y se aplican a lo largo de todo el proceso del software. Estas técnicas tendrán como objetivo certificar la concordancia entre la aplicación y los requisitos funcionales explícitamente establecidos, los estándares de desarrollo documentados y las características implícitas que se esperan de cualquier software desarrollado profesionalmente.

En la realidad la SQA comprende tareas llevadas a cabo por dos tipos de personal, los desarrolladores de software y un grupo SQA que ayuda al equipo de desarrollo para conseguir un producto de alta calidad. En nuestro caso, seremos nosotros mismos, desarrolladores del proyecto, los que asumamos las funciones del equipo SQA.

Nuestro control de calidad comprenderá una serie de inspecciones, auditorías y pruebas a lo largo de todo el proceso del software para asegurar que nuestro producto cumple con los requisitos que le hayan sido asignados.

Así, la principal medida de garantía de calidad de que dispondremos serán las Revisiones Técnicas Formales (RTFs) que nos permitirán descubrir errores, verificar que el software cumple con los requisitos, conseguir que el software sea desarrollado de manera uniforme y que nuestro proyecto sea más mantenible.

Cada RTF se producirá a partir de una reunión previamente planificada. Así pues, cada vez que se finalice una determinada tarea relevante en el proyecto (como la codificación o diseño de alguna parte del software) se realizará una RTF a la que acudirá el equipo al completo. Dicha reunión deberá ser preparada de antemano y no durará en ningún caso más de dos horas. En ella se realizará una revisión del trabajo realizado, además de un informe sumario de la reunión que comprenderá la fecha en la que la revisión tuvo lugar, el material revisado y el resultado de dicha revisión (modificaciones propuestas, por ejemplo). En la mayoría de las RTFs contemplamos que esté presente el director del proyecto, de forma que podamos conocer su opinión del material realizado hasta el momento y de las conclusiones obtenidas.

2.7.2 Gestión y control de cambios

Llevaremos a cabo una gestión de la configuración software de nuestro proyecto para realizar el control de cambios.

En principio, solo contemplaremos cambios producidos a partir de la detección de un fallo en el comportamiento del sistema. Por tanto, no se esperan cambios en la especificación de requisitos o en el modelado de la aplicación en funcionalidades que ya estén implementadas.

Tampoco habrá restricciones presupuestarias que puedan propiciar cambios en nuestro proyecto, dado el carácter académico del proyecto.

Sin embargo, una mala planificación del trabajo o la baja de un miembro del equipo si puede dar lugar a un proceso de cambio, tal y como ha quedado descrito en el apartado 2.3.2 de este Plan de Proyecto Software.

Definiremos el concepto de *línea base* como un parte del proyecto que ya ha sido revisada de manera formal en una RTF y que servirá de base para un desarrollo posterior. Dicha línea base sólo podrá ser cambiada mediante procedimientos formales de control de cambios.

Así mismo, definimos ECS (elemento de configuración software) como cada una de estas piezas de información generada durante los procesos de la ingeniería del software.

Una vez dadas estas dos definiciones, nuestra gestión de la configuración software se centrará en cinco actividades fundamentales:

(a) Identificación de ECSs: Para controlar y gestionar los ECSs emplearemos un enfoque orientado a objetos. Identificaremos los objetos como unidades de texto creadas por un miembro del equipo durante el proceso de IS (el plan de proyecto, una parte de la SRS, una parte del diseño, casos de prueba, código implementado, etc.).

(b) Control de versiones: El control de versiones nos permitirá gestionar la versión del sistema, que a su vez vendrá identificada por las versiones de los ECSs. Dicho control se gestionará de forma automáticamente mediante la herramienta SVN Subclipse, tal y como ha quedado descrito en el apartado 5.2 de este Plan de Proyecto Software.

(c) Control de cambios: Los cambios, antes de que cualquier ECS se convierta en línea base, se llevarán a cabo sin procedimientos SQA. Sin embargo, una vez cerrada una línea base, serán necesarios procedimientos formales para realizar cambios sobre dicho ECS.

Nuestro mecanismo para el control de cambios será el siguiente: si cualquier miembro del equipo desea realizar un cambio sobre una parte del proyecto ya revisada, deberá comunicarlo a los demás miembros del grupo en una de nuestras reuniones. Seguidamente se estudiará el cambio y sus posibles repercusiones sobre el proyecto, siempre de acuerdo el director del proyecto. En caso de que el cambio sea aprobado se procederá a implementarlo. Si el cambio es de una magnitud considerable, se procederá a crear una nueva versión del ECS que va a sufrir las modificaciones.

(d) Auditorias de configuración software: Debemos realizar un seguimiento exhaustivo de los cambios que realicemos en nuestro proyecto y la manera que utilizaremos para asegurar que el cambio se ha llevado a cabo correctamente será mediante las auditorias de configuración software. En las RTFs, nos preocuparemos de la corrección técnica del cambio y mediante las auditorias nos preocuparemos de si se han hecho los cambios especificados, se han incorporado modificaciones adicionales, se ha llevado a cabo una RTF, se han seguido correctamente los estándares de IS, se han seguido procedimientos de gestión de configuración software para gestionar los cambios, etc.

(e) Informes de estado: No se llevarán a cabo informes de estado. El uso de herramientas de control de versiones como SVN nos provee automáticamente de informes de estado sobre la fecha, el autor y el contenido de los cambios realizados en la aplicación.

2.8. BIBLIOGRAFÍA

Consultar la bibliografía al final del documento.

CAPÍTULO 3
ARQUITECTURA E IMPLEMENTACIÓN

3.1 INTRODUCCIÓN

3.1.1 Propósito

El diseño e implementación de un sistema software usando metodología orientada a objetos exige una distribución óptima de roles y responsabilidades entre los componentes del sistema. En cuanto dicha aplicación adquiere un tamaño considerable estos requerimientos se vuelven aun más necesarios si cabe, por lo que resulta imprescindible estructurar el código de forma que éste sea mantenible, escalable y entendible, es decir, aumentando la cohesión y reduciendo el acoplamiento entre las partes del mismo. Debido a este motivo se hace por tanto indispensable el uso de arquitecturas software de alto nivel.

Una arquitectura multicapa particiona todo el sistema en distintas unidades funcionales: cliente, presentación, lógica de negocio, integración, etc. Esto asegura una división clara de responsabilidades y hace que el sistema sea más flexible al cambio. El uso de arquitecturas de varios niveles conlleva una serie de ventajas e inconvenientes.

Ventajas:

- Se puede modificar cualquier capa sin afectar a las demás.
- Integración y reusabilidad.
- Encapsulación
- Distribución
- Particionamiento
- Mejora de la fiabilidad
- Manejabilidad
- Incremento en la consistencia y en la fiabilidad.
- Soporte para múltiples clientes.

- Desarrollo independiente.
- Desarrollo rápido.
- Empaquetamiento
- Configurabilidad

Inconvenientes:

- Mayor complejidad arquitectónica.
- Posible pérdida de rendimiento y escalabilidad.
- Riesgos de seguridad.
- Gestión de componentes.

3.1.2 Metodología de trabajo

Consideramos que el perfil de nuestra aplicación encajaba perfectamente en las características de un modelo multicapa, sin embargo, a la hora de tomar esta decisión, se realizó un análisis para definir de forma aun más concreta cuantos niveles iba a requerir nuestro sistema software y que funcionalidades debería llevar a cabo cada uno. Para ello, partimos de una definición estandarizada de una arquitectura multicapas de tres niveles:

- La capa de **presentación** expone los servicios de la capa de "lógica" a los usuarios. Sabe cómo procesar una petición de cliente, cómo interactuar con la capa de "lógica", y cómo seleccionar la siguiente vista a mostrar.
- La capa de **lógica** proporciona los servicios del sistema.
- La capa de **persistencia** es la responsable de la comunicación con recursos y sistemas externos.

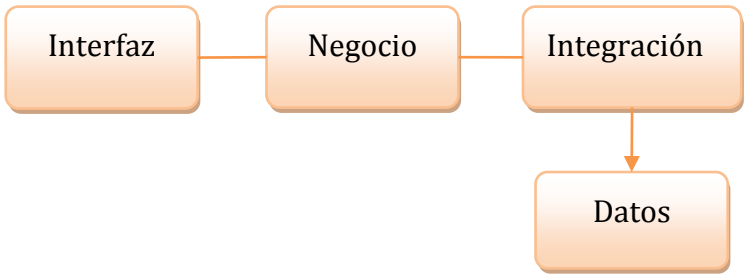


Figura 1 Diagrama de Capas

Aunque no suela estar definido de forma explícita, esta clase de arquitecturas constan de dos niveles más, el de **clientes**, que representa a todos los clientes o dispositivos del sistema que acceden al mismo, que estaría sobre la capa de presentación, y la capa de **recursos** que contiene los datos del negocio y recursos externos. Estaría situada bajo la capa de integración.

En nuestro caso, CPMP constaba de todos los requisitos arriba mencionados. Por un lado, la división en capas era evidente. Teníamos una serie de interfaces gráficos que ofrecían los diversos servicios de la aplicación al usuario, un conjunto de lógica encargada de parsear los catálogos, gestionar los eventos e interpretar los scripts, y unos datos que exigían persistencia. En la siguiente figura, un diagrama de paquetes, se podrán entender mejor estas palabras.

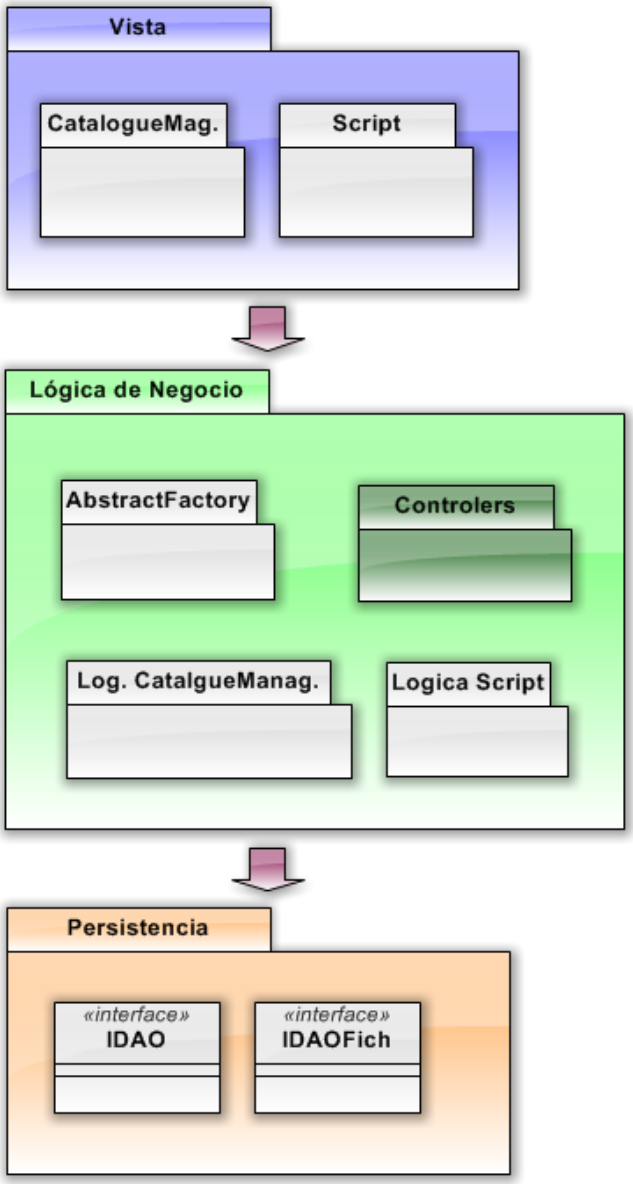
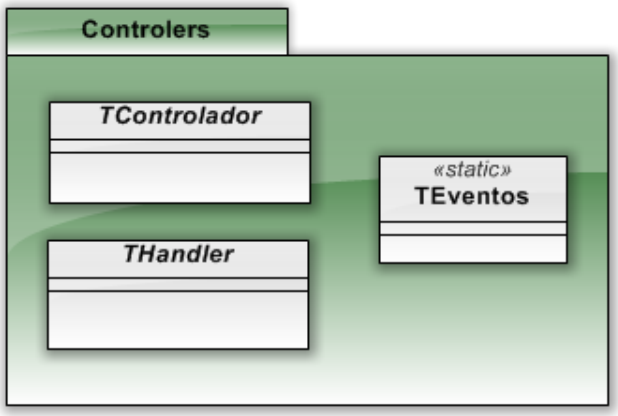


Figura 2 Diagrama de Paquetes



Por otro lado, la mantenibilidad, la fiabilidad, el particionamiento, la manejabilidad o la mejora del rendimiento eran factores indispensables en la aplicación y su desarrollo. El uso de una arquitectura de varios niveles no solo permitió trabajar de forma separada e independiente, sino que además ha dado como resultado un código muy mantenible, estable y fácilmente depurable. La gestión del cambio ha resultado ser una tarea sencilla puesto que, al estar dividido el código en capas independientes, no existieron ramificaciones inesperadas a partir de dichos cambios.

Antes de entrar en los detalles de la implementación, sería recomendable volver a recapitular todas las funcionalidades que implementará este sistema. Para ello se muestra en la siguiente figura el diagrama de casos de usos de CPMP:

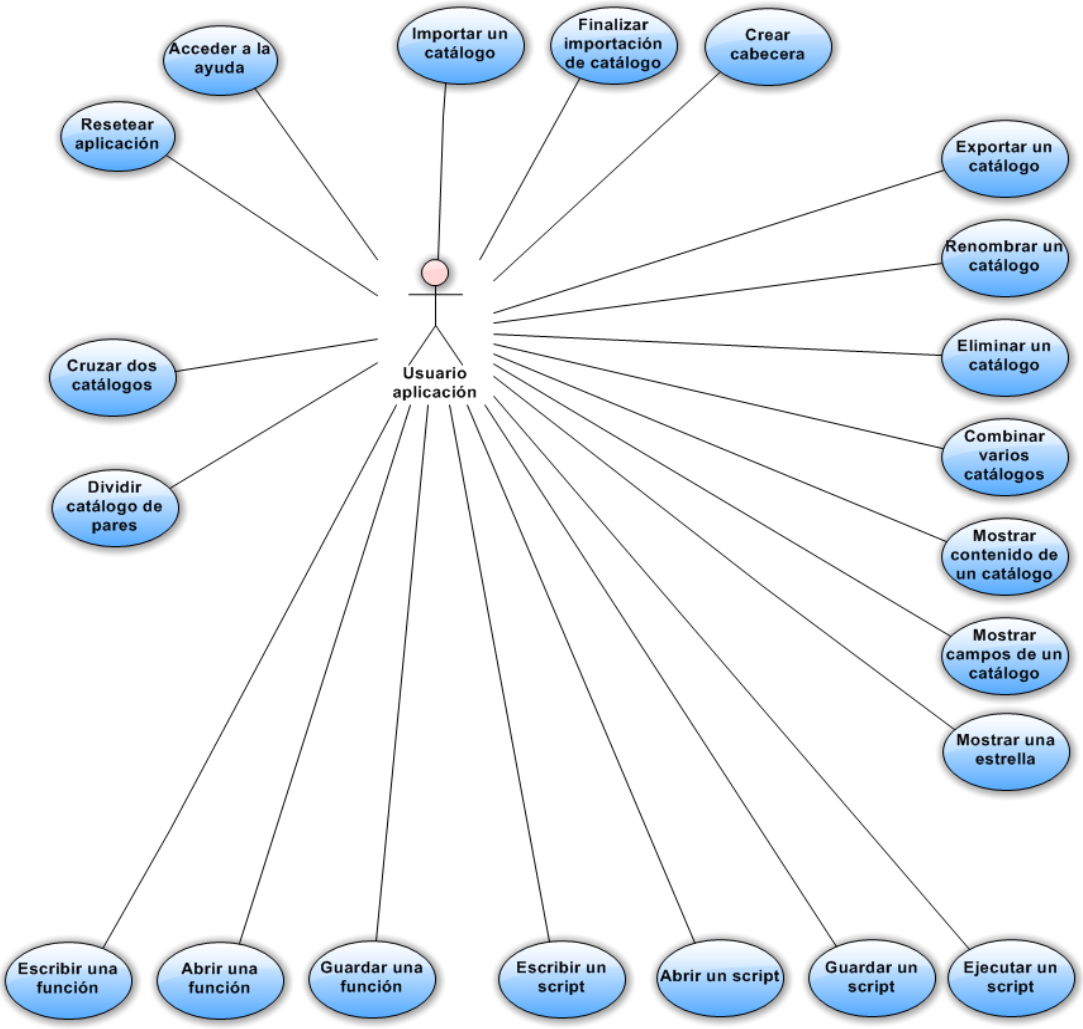


Figura 3 Diagrama de Casos de Uso

3.2 REVISIÓN DE ESPECIFICACIÓN DE REQUISITOS SOFTWARE

Los requisitos definidos al principio de la aplicación sólo han sufrido un par de variaciones. La funcionalidad de descargar catálogos de VizieR fue descartada inicialmente porque la dificultad que entrañaba poner en marcha el servicio web de VizieR no compensaba el resultado. VizieR ya dispone de un interfaz online en el que el usuario puede descargar los catálogos ajustando toda clase de parámetros, por lo que nosotros no íbamos a ofrecer al usuario nada que él ya no pudiera hacer, probablemente, de forma más eficiente y completa a través de la página web.

El conjunto de requisitos también fue modificado para añadir la funcionalidad de crear cabecera. De forma breve, este caso de uso da aun más flexibilidad a la aplicación al permitir crear, simplemente a partir de cualquier tipo de datos, un catálogo completo que importar a la aplicación. Esto, en términos más globales, supone muchísima más potencia y alcance para la aplicación, ya que nos permite crearnos por ejemplo nuestros propios catálogos, o si en un futuro cambian el formato de los catálogos alojados en VizieR, la aplicación no quedará obsoleta, ya que simplemente tendremos que crear una nueva cabecera para estos catálogos que sea compatible con CPMP.

Crear cabecera VizieR

- **Función:** Crear cabecera VizieR.
- **Prioridad:** Media.
- **Estabilidad:** Media.
- **Descripción:** Crea un nuevo catálogo, en el disco del usuario, con una cabecera VizieR válida a partir de un fichero que contiene únicamente estrellas.
- **Entrada:** Selección de un fichero ya existente mediante ratón. El usuario deberá insertar por ratón/teclado los datos de la nueva cabecera. Los campos insertados y sus tipos deben encajar con las estrellas contenidas en el catálogo, de lo contrario, se producirá un error. Deberán existir como mínimo los campos RAJ2000 y DEJ2000.

- **Salida:** Nuevo catálogo con una cabecera VizieR válida en forma de fichero de texto.
- **Origen:** Usuario del sistema.
- **Destino:** Sistema.
- **Necesita:** Fichero binario o de texto con un listado de estrellas.
- **Acción:** Selección de un fichero que contenga únicamente estrellas. Se pedirá al usuario introducir los datos necesarios para crear un nuevo catálogo con la cabecera VizieR correspondiente.
- **Precondición:** El nombre de la nueva tabla no debe existir previamente en el sistema.
- **Postcondición:** Actualización de las tablas de la base de datos que guardan el estado del sistema. Añadida una nueva tabla con el catálogo correspondiente.
- **Efectos laterales:** -

3.3. IMPLEMENTACIÓN

A continuación, se presenta un desglose más extenso de las decisiones de diseño e implementación que fueron realizadas durante el desarrollo del proyecto capa a capa.

3.3.1. Capa de presentación

Esta capa es la encargada de la interacción con el usuario, más concretamente, la encargada de mostrarle la información y los servicios que ofrece el sistema, además de capturar los eventos externos a dicho sistema.

Hemos implementado esta capa usando patrones auxiliares como el "Singleton", que nos proporciona una única instancia de un objeto y un único acceso global al mismo, y que también hemos usado en los otros niveles.

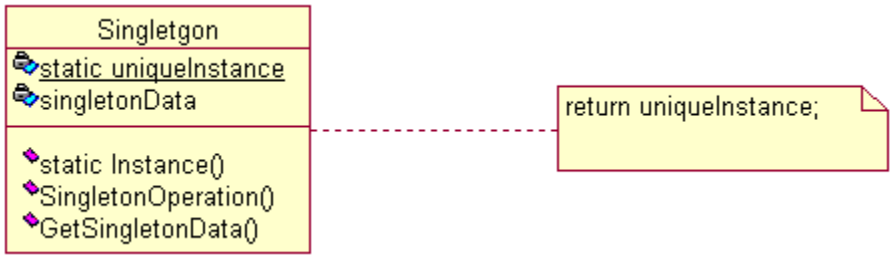


Figura 4 Estructura del Patrón Singleton

Sin embargo, el principal patrón de diseño usando en esta capa ha sido el "MVC", modelo vista controlador, que divide una aplicación interactiva en tres componentes:

- El modelo contiene la funcionalidad básica y los datos (serían nuestras capas de integración y negocio)
- Las vistas muestran y recogen información del usuario.
- El controlador media entre vistas y modelo.

De sus dos variantes, activa y pasiva, nosotros utilizaremos la segunda.

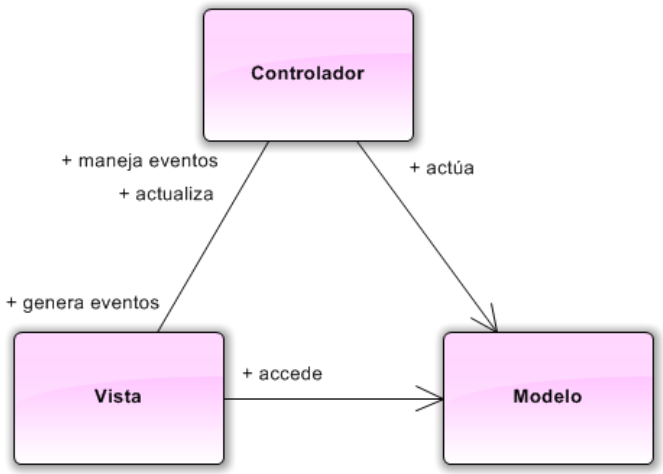


Figura 5. Participantes en el MVC. Modelo Pasivo

Consideramos pertenecientes a nuestra capa de presentación tanto la Vista como el Controlador.

El usuario intéractua con la aplicación a través de las diversas interfaces graficas (GUI´s) que están puestas a su alcance, ya sea en el módulo de gestión de catálogos o en el de desarrollo de scripts. Normalmente el usuario demanda un

servicio, como por ejemplo, obtener el listado de catálogos, las estrellas dentro de un catálogo, o la ejecución de un script. Estas peticiones de servicios son consideradas como eventos que recoge y administra un Controlador.

A la hora de diseñar el Controlador, tuvimos varias alternativas. Por un lado, estaba la opción de implementar varios controladores distintos, uno por cada módulo de la aplicación. La otra alternativa era utilizar un controlador único codificado como patrón Singleton. Se optó por esta última opción ya que, por un lado, el número de módulos claramente diferenciados de los que iba a constar el sistema era únicamente dos, y además, algunas funcionalidades estaban compartidas por ambos. De este modo, el controlador no ejercía únicamente como un gestor de eventos, sino que además se convertía en una fachada o puerta de entrada única por la que tenían que pasar todas las demandas de servicios que se quisieran realizar a la capa de negocio.

Para tal tarea, el Controlador fue implementado con un método "acción", que recibe un objeto de la superclase Object para poder procesar cualquier tipo de dato insertado por el usuario, y un código numérico que hace referencia a una entrada dentro de un diccionario de eventos. De esta forma, el controlador es capaz de detectar el evento y direccionar la demanda de servicios de forma adecuada. En ocasiones, ha sido necesario llevar a cabo algún preprocesamiento sobre los datos recibidos, aunque era una práctica que intentamos evitar en la medida de lo posible.

Una vez realizada la funcionalidad requerida, y puesto que estábamos utilizando un MVC pasivo, el controlador debía actualizar la Vista con la cual había interactuado el usuario.

Aquí surgió un nuevo problema de diseño, ya que en muchos casos, la demanda de servicio que realizaba el usuario suponía efectuar cambios en vistas de distintos módulos. Por ejemplo, el usuario podía demandar el borrado de un catálogo, y este cambio debía reflejarse no solo en la Vista del módulo de gestión de catálogos, sino también dentro del listado que poseía la Vista de desarrollo de scripts, siempre y cuando ésta se hayase activada.

Es decir, teníamos dos problemas, por un lado, actualizar dos módulos diferentes, y por otro lado, que uno de los módulos pudiese no estar activado, es decir, no existir sus instancias dentro de la aplicación.

Es por ello que el equipo de desarrollo tomó la opción de añadir al conjunto de clases que conformaban la capa de presentación un manejador de actualizaciones, basándonos en el concepto del patrón Chain of Responsability. El patrón de diseño Chain of Responsibility permite establecer una cadena de objetos receptores a través de los cuales se pasa una petición formulada por un objeto emisor. Cualquiera de los objetos receptores puede responder a la petición en función de un criterio establecido. En nuestro caso, la idea era propagar las actualizaciones por toda la capa de presentación.

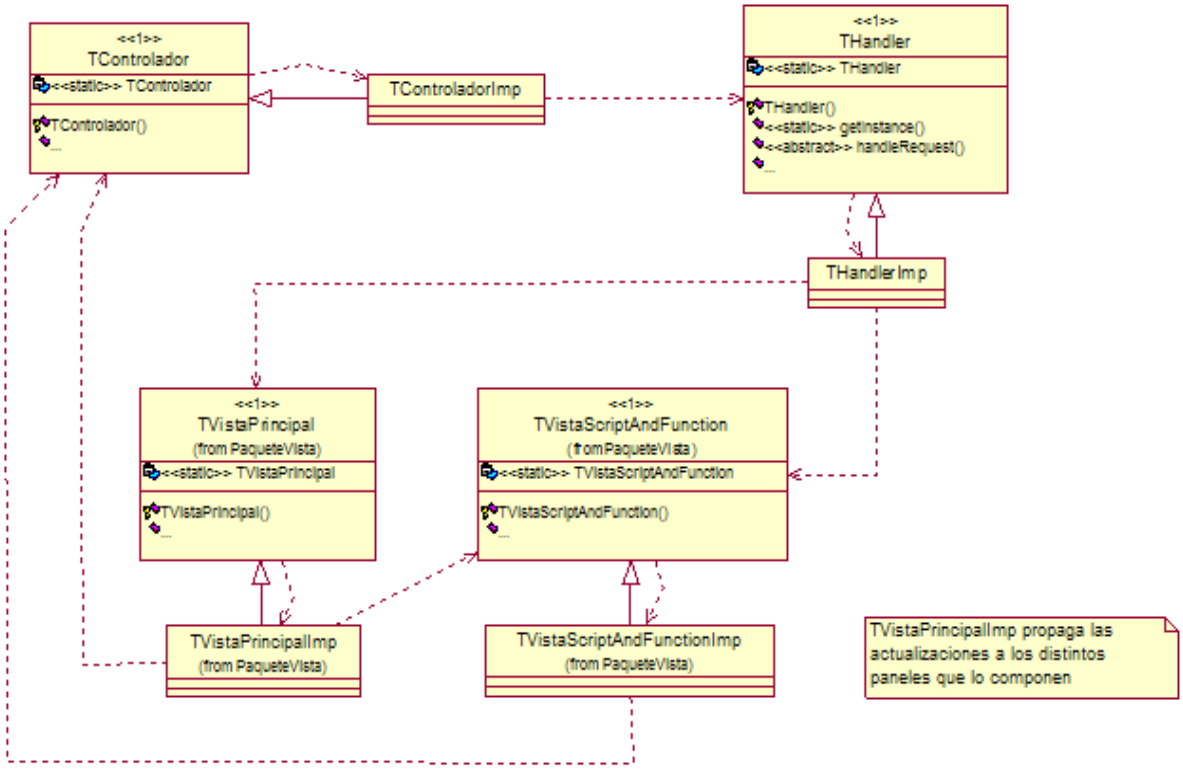


Figura 6 Diagrama de Clases del
THandler, TControlador y las vistas principales

Este manejador de actualizaciones debía por tanto contener información acerca de qué vistas estaban activas en cada momento, por lo que debía existir una instancia única y activa de la clase durante todo el tiempo de vida de la aplicación,

y por último, propagar por cada una de estas vistas todas las actualizaciones, siempre en función del servicio demandando por el usuario.

Recapitulando un poco, teníamos tres actores principales en esta capa, por un lado, un conjunto de vistas o interfaces gráficas para interactuar con el usuario, por otro, un Controlador encargado de capturar y gestionar los eventos generados a raiz de las demandas de servicios realizadas por el usuario, y por último, un manejador de actualizaciones encargado de propagar allá donde es necesario el resultado de dichas demandas de servicios.

Una vez definida de forma concisa el comportamiento de dos de estos actores principales, era importante ver como se iban a implementar cada una de estas interfaces gráficas a las que ya hemos hecho referencia, y sobre todo, como iban a interactuar entre ellas. Sabíamos que debíamos tener al menos dos interfaces gráficas, una para la gestión de catálogos y otra para el desarrollo de scripts, pero solo una, la primera, iba a estar activa en la aplicación. Sabiamos también que estas vistas iban a ser actualizadas por un objeto externo. Las primeras decisiones de diseño fueron por tanto empezar a trabajar a partir de la superclase JFrame y que estas GUI´s implementasen la interfaz IVista, la cual contenía unicamente el método “actualiza”.

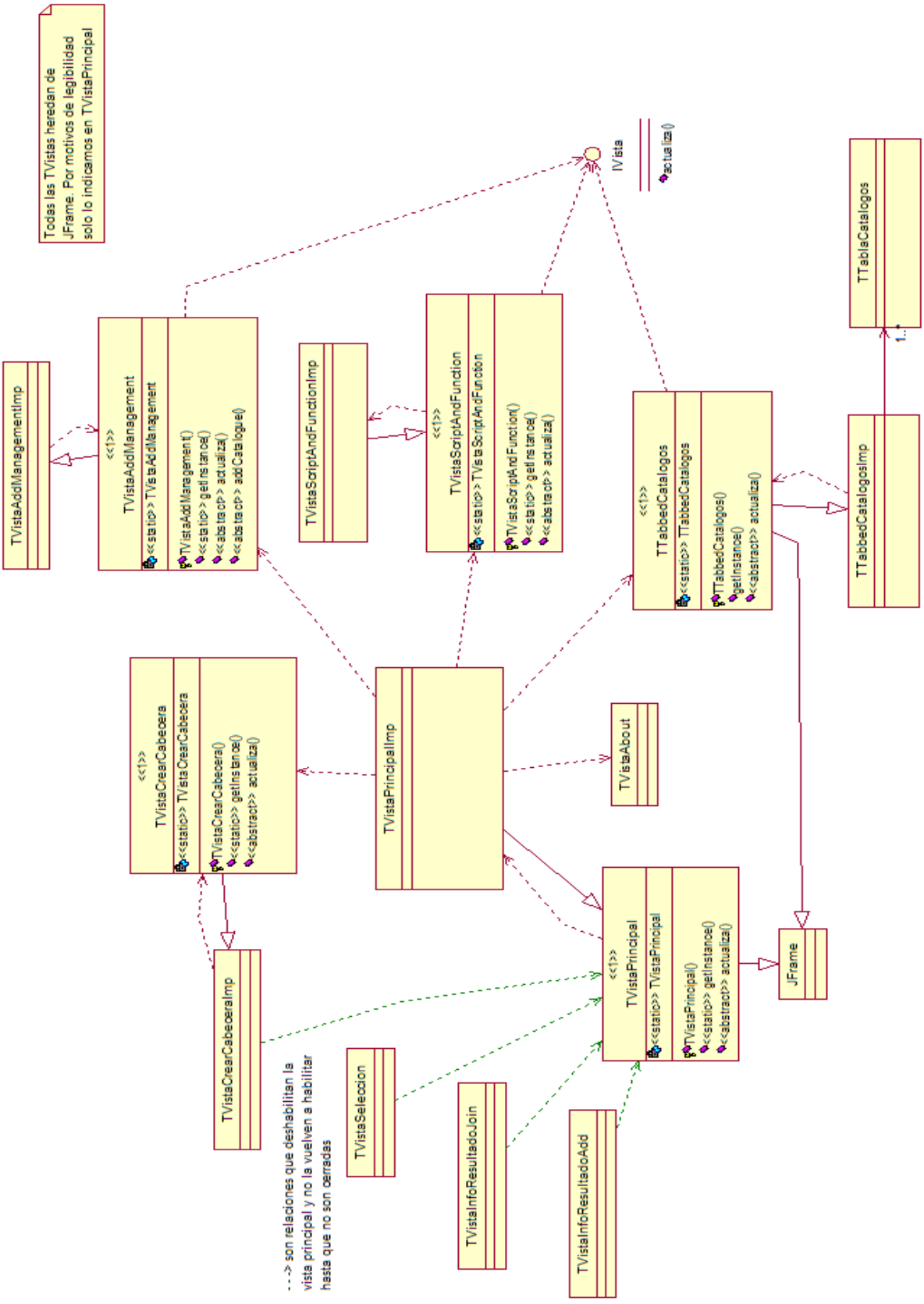


Figura 7 Diagrama de Clases de las Vistas

La interfaz gráfica para el desarrollo de scripts debía ser una única ventana con los elementos necesarios para tal tarea, como un panel de escritura, una consola que mostrase el resultado de la ejecución de dichos scripts, y un panel lateral que contuviera el listado de los catálogos incluidos en la aplicación. A priori, no necesitaba de más trabajo que el requerido para la implementación de los distintios componentes del panel o para añadir la posibilidad de usar atajos de teclado y menús contextuales sobre dichos componentes.

El único problema en este módulo se encontró a la hora de actualizar la consola. La idea inicial era que a medida que cada instrucción se fuera ejecutando, se mostrase información sobre el resultado de esta. Desafortunadamente, esta información aparecía en pantalla de golpe. Fue necesario llevar a cabo una pequeña documentación sobre el comportamiento y actualización de los elementos de Swing para obtener el resultado deseado.

Por otra parte, la interfaz de gestión de catálogos requería mostrar mucha y muy diversa información, como un listado de catálogos o las estrellas dentro de cada uno de ellos. Prácticamente se podía decir que era necesaria una Vista por cada caso de uso.

Nuevamente, nos encontrábamos ante una disyuntiva de diseño. Podíamos optar por representar cada uno de estos casos de uso con una ventana nueva e independiente, o usar distintos paneles que iríamos rotando dentro de la misma ventana.

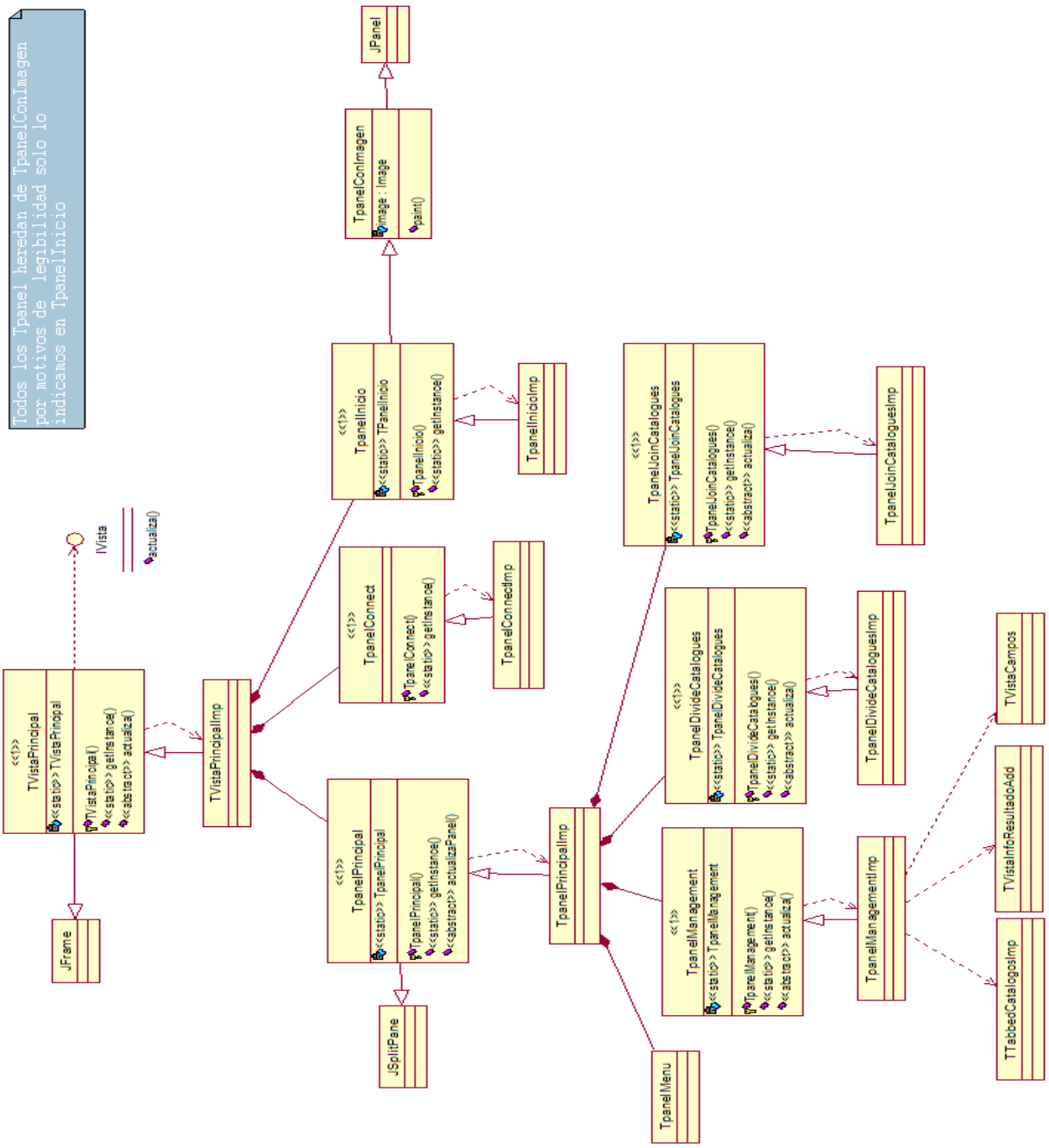


Figura 8 Diagrama de Clases de los Paneles

En este caso, se optó por una solución mixta. Era bastante razonable que un usuario pudiese tener abierto más de un catálogo al mismo tiempo para observar las estrellas contenidas en él. El proceso de navegar constantemente de un cátalogo a otro con el consiguiente trabajo de volver a encontrar la estrella

deseada y sus valores, podía llegar a ser realmente tedioso. Si el usuario tuviera la posibilidad de tener en pantalla varios catálogos abiertos al mismo tiempo, este problema se vería solucionado. Lo mismo sucedía con el listado de campos, el cual podía ser util tener presente mientras se utilizaban otras funcionalidades de la aplicación. Para el resto de GUI´s, elegimos el uso de paneles salvo en un solo caso, el de la funcionalidad de "crear cabecera", que fue un requisito pedido por el director del proyecto a posteriori. Una vez verificado que este caso de uso proporcionaba a la aplicación una mejora realmente notable en la flexibilidad para trabajar con todo tipo de formatos de catálogos, se optó por implementarla como una ventana aparte de tal forma que se evitase tocar en la medida de lo posible el trabajo ya realizado (lo que en Ingeniería del Software se conoce como *linea de base*).

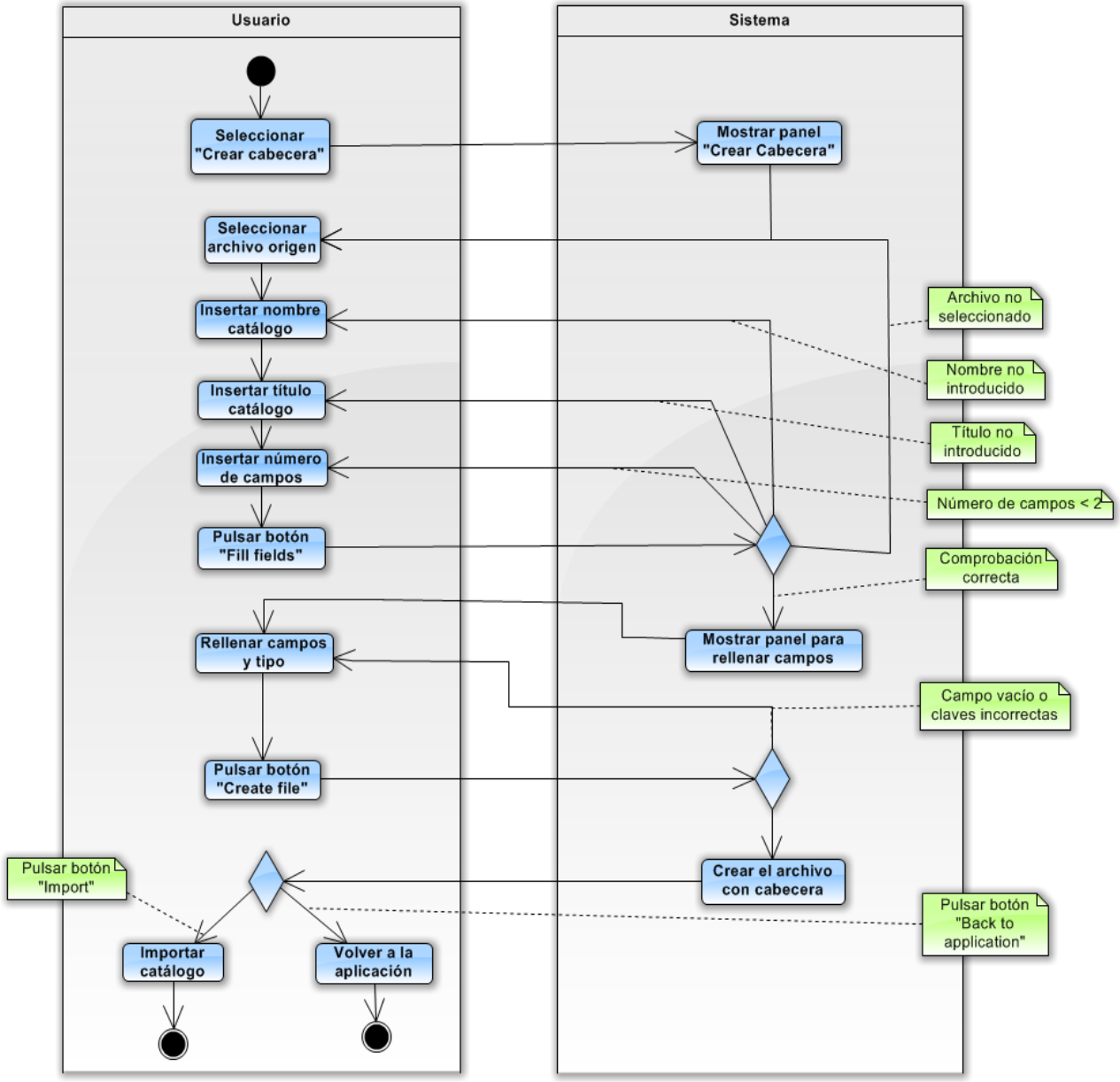


Figura 9 Diagrama de Actividad Crear Cabecera

Por último, también eran necesarias una serie de ventanas de menos dimensiones y sin funcionalidad alguna que se encargasen de mostrar al usuario los informes con los resultados de las operaciones efectuadas. Por ejemplo, si el usuario insertaba un catálogo, al final del proceso se le mostraría una pequeña ventanita informándole del número de estrellas insertadas, el número de duplicadas, y el tiempo transcurrido en realizar el proceso.

Llegaba finalmente el momento de pensar que apariencia queríamos que tuviesen las distintas intefaces. Por un lado, estaba claro que se iba a utilizar la API Swing de Java, que nos proporcionaba una potencia gráfica a priori suficiente para nuestros propósitos. Sin embargo, el uso de esta API se acabó convirtiendo en una labor tediosa y problemática que no ofrecía los resultados esperados. Nuestro objetivo era evitar el uso de cualquier plugin o editor de GUI´s integrado como los que se pueden encontrar en entornos de desarrollo como Eclipse o NetBeans, ya que el código que generan estos editores suele ser totalmente inmantenible, por lo que generamos todos las interfaces gráficas manualmente. Sin embargo, ninguno de los miembros del grupo disponía de demasiada experiencia en el uso de las funciones y elementos incluidos en Swing más allá de los conocimientos básicos, lo que se traducía en interfaces gráficas de aspecto un tanto pobre que no hacían justicia con el laborioso trabajo que la aplicación llevaba por debajo.

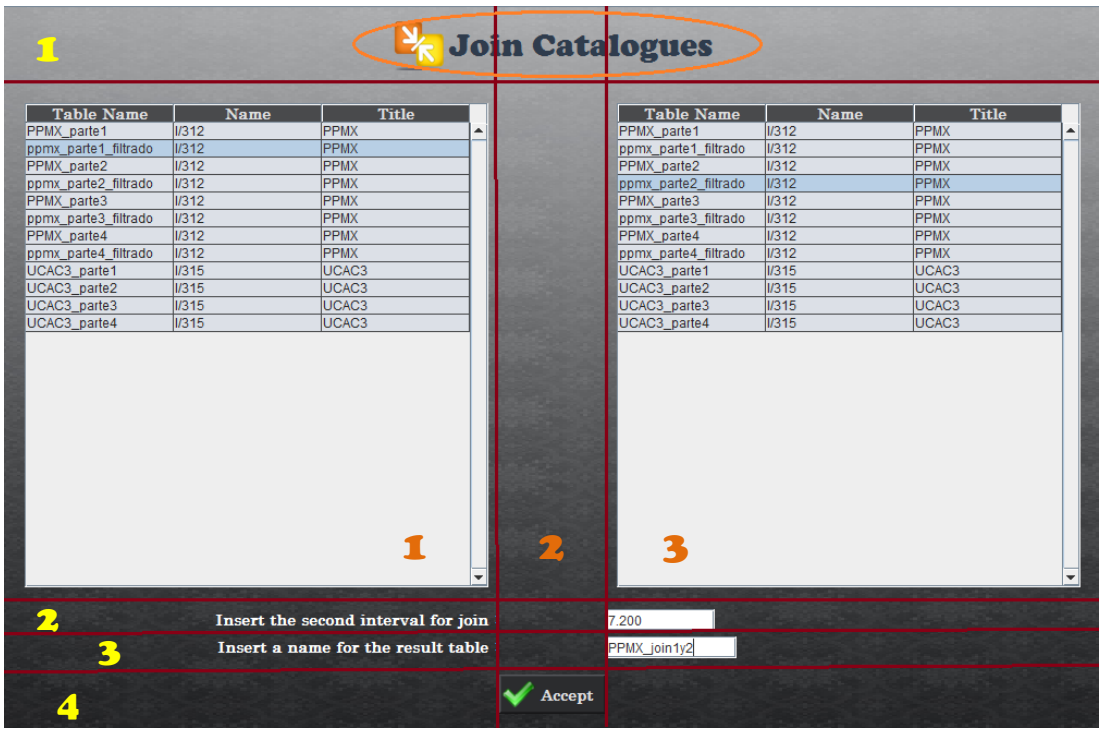


Figura 10 Ejemplo de uso de GridBagLayout

Aunque al director del proyecto le parecía más que suficiente el trabajo realizado al respecto pues permitía un correcto uso de las funcionalidades de la aplicación, el equipo de desarrollo decidió realizar una mejora sustancial de estos interfaces. Para ello, primero analizamos el aspecto gráfico de una decena de

aplicaciones realizadas con Java para conocer el alcance y las posibilidades que esta plataforma nos ofrecía, y una vez recopiladas una serie de buenas ideas, procedimos a documentarnos sobre como se implementaban los elementos que queríamos añadir a nuestra aplicación. Gran parte de este trabajo se consiguió mediante el uso del GridBagLayout de Swing, el cual nos permitía compartimentar cada panel en celdas de distintas dimensiones.

La implementación del comportamiento de todos estos paneles y ventanas, más allá de todo lo ya mencionado, no entrañó excesiva dificultad. Las ventanas captan las peticiones del usuario, recogen los datos insertados en caso de haberlos, y se realiza una llamada al Controlador para que gestione el evento. Existió, no obstante, una funcionalidad que resultó ser bastante más dificultosa de lo esperado: mostrar el contenido de los catálogos.

Por un lado, cada catálogo tiene una serie de campos diferentes, por lo que la generación en pantalla de la tabla que debía mostrar las estrellas se debía realizar dinámicamente a partir de un listado con los campos del catálogo en cuestión. Esto, aunque reducía el rendimiento de la aplicación, nos permitía manejar de forma homogénea cualquier tipo de catálogo. Aun así, esto no dejaba de ser un simple contratiempo comparado con el inconveniente que se avecinaba: el tamaño de los catálogos.

Como ya hemos comentado en reiteradas ocasiones, los catálogos pueden llegar a ocupar varios gigas de memoria en el peor de los casos. Sin embargo, en nuestros ejemplos de prueba trabajábamos por comodidad con catálogos truncados a un número muy reducido de estrellas, que eran fácilmente mostradas al usuario por pantalla. El problema llegó cuando nos dimos cuenta de que la aplicación final no debía mostrar un listado de 100 estrellas sino millones de ellas. Esto suponía dos problemas, primero, que no era en absoluto razonable mostrar una tabla con cientos de miles de entradas forzando al usuario a recorrerla mediante el uso del scroll, y segundo, que llegado el caso, tampoco se podría llevar a cabo esta opción por la falta de memoria de la Máquina Virtual de Java. La situación requería un análisis concienzudo, ya que por supuesto no todos los

catálogos ocupaban semejante tamaño, es decir, debíamos encontrar nuevamente una solución homogénea para cualquier tipo de catálogo.

Era necesario pues, por un lado, optimizar el uso de memoria al máximo, y por otro, encontrar una forma de representar la información contenida en cualquier tipo de catálogo. Para el primer caso, no quedó más remedio que acudir a la documentación de referencia de la Máquina Virtual para comprender como funcionaba la gestión de memoria y el recolector de basura. Mediante un trabajo de limpieza de referencias y algunas llamadas al "Runtime System" para que éste pusiese especial hincapié en liberar memoria en las zonas críticas del código, conseguimos mejorar sustancialmente el uso de los recursos del sistema.

Por último, para mostrar los catálogos, hicimos un uso de un buffer un tanto especial. Este buffer consultaba cuantas entradas tenía el catálogo, y en función de este número, optaba por mostrarlas todas de golpe o poco a poco. Si era necesario mostrarlas poco a poco, se iban generando una serie de intervalos de tal forma que el buffer pudiese cargar y mostrar siempre el máximo número de estrellas sin pasarse de los límites que tenía asignados. Estos intervalos eran generados a partir de las coordenadas primarias de las estrellas, su ascensión recta, y su declinación (solo de la AR), medidas en grados decimales. El usuario de esta forma podía navegar adelante y atrás por los intervalos, teniendo siempre a su disposición información sobre el intervalo de grados en el que se hallaba, el numero de estrellas totales del catálogo, el número de estrellas mostradas por pantalla, y una entrada de formulario para que pudiese introducir la longitud del próximo intervalo a mostrar, siempre y cuando no se superasen los límites del buffer. En este caso, se aplicaba un algoritmo que iba reduciendo la longitud del intervalo hasta que fuera posible su impresión en pantalla.

3.3.2. Capa de lógica

La capa de la "lógica de negocio" contiene los objetos y servicios de la aplicación. Recibe peticiones de la capa de presentación, procesa la lógica de negocio basada en estas peticiones, y finalmente desencadena los accesos a los recursos de la capa de datos.

Como ya hemos comentado anteriormente, la capa de lógica o de negocio (usaremos ambos nombres indistintamente) es accedida mediante el Controlador, el cual se encarga al mismo tiempo de proporcionar los datos que se van a necesitar para realizar la lógica de la aplicación. Esta información, que el cliente del sistema debe o bien visualizar o bien introducir, se encapsula en un objeto que será el elemento de comunicación entre capas. Este tipo de diseño, conocido como patrón Transfer, nos permite independizar las capas de tal manera que estas no puedan tener conocimiento de la representación interna de las entidades de nuestro sistema.

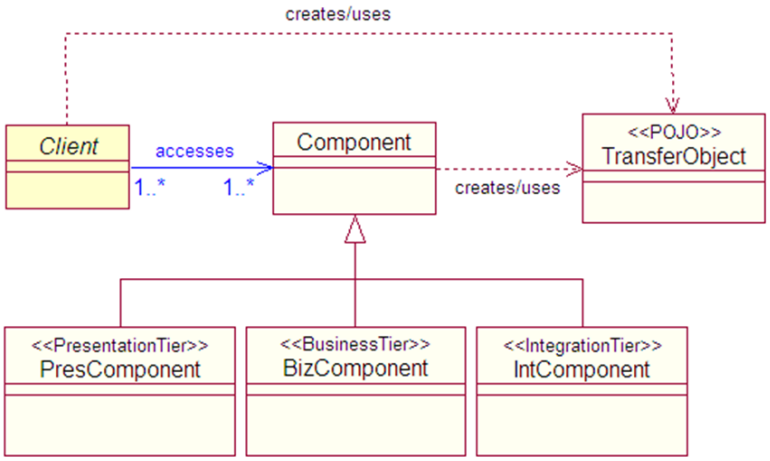


Figura 11 Estructura del Patrón Transfer

En nuestro caso concreto, estos Transfers contendrán la información incluida en el apartado 1.3.1.1 de la Especificación de Requisitos Software, como la representación de un catálogo en función de su nombre, su descripción y su nomenclatura VizieR, o el nombre y el tipo de cada uno de los campos que representan a una estrella.

El problema en este punto, y uno de los principales retos de la aplicación, se hallaba exactamente en dichas estrellas. Era requisito imprescindible que CPMP pudiera manejar estrellas con cualquier tipo de formato. Recordemos que no existe estándar alguno a la hora de representarlas, es decir, en algunos catálogos las estrellas pueden estar simplemente codificadas con sus coordenadas, mientras que en otros, aparte de las coordenadas podrían existir incluso decenas de campos más como el brillo, la velocidad con la que se desplazan o cualquier otro tipo de valor

alfanumérico. Esto hacía imposible la creación de cualquier tipo TEstrella que nos ayudase a manejar la información de forma encapsulada en las distintas capas.

A la hora de solventar este escollo, se manejaron varias opciones que acabaron siendo descartadas o bien por su complejidad, o bien por su coste en rendimiento. Por ejemplo, se valoró la posibilidad de implementar clases dinámicas que fueran declaradas en base a la información contenida en los ficheros de texto que conformaban los catálogos. Se valoró también la posibilidad de crear una clase TEstrella que contuviera ciertos campos mínimos, dejando el resto de valores encapsulados en un array polimórfico también contenido como atributo de la clase. Ninguna de estas opciones era demasiado tentadora, bien porque implicaba una complejidad excesiva respecto a la funcionalidad que aportaba a la aplicación, o bien porque limitaba demasiado nuestra flexibilidad al obligarnos a definir unos campos mínimos que podrían no ser útiles, ya sea en otros catálogos, o en un futuro.

Finalmente, el tipo TEstrella quedó sin definir y se optó por encapsular las estrellas totalmente dentro de arrays polimórficos. El diseño de la aplicación quizás no fuera tan elegante, pero las ganancias en simplicidad y flexibilidad merecían la pena. El uso combinado de estos arrays polimórficos, encargados de encapsular las estrellas de cada catálogo, junto con un listado de los campos y sus tipos disponible desde el mismo momento en que un catálogo era insertado, solventaba todas nuestras necesidades.

Una vez concluida la forma de representar los datos, llegaba el momento de implementar cada uno de los casos de uso de la aplicación, y para ello, partíamos de una división evidente entre las funcionalidades de los dos módulos.

Comenzamos esta tarea con la parte que a priori era más sencilla, la de gestión de catálogos. Algunos miembros del equipo ya tenían cierta experiencia en el desarrollo de sistemas de información básicos, por lo que funcionalidades como borrar o actualizar, ya de por sí bastante simples, no entrañaron dificultad. En la siguiente figura se muestra un ejemplo de una de dichas funcionalidades básicas, la de unir dos catálogos compatibles en uno solo.

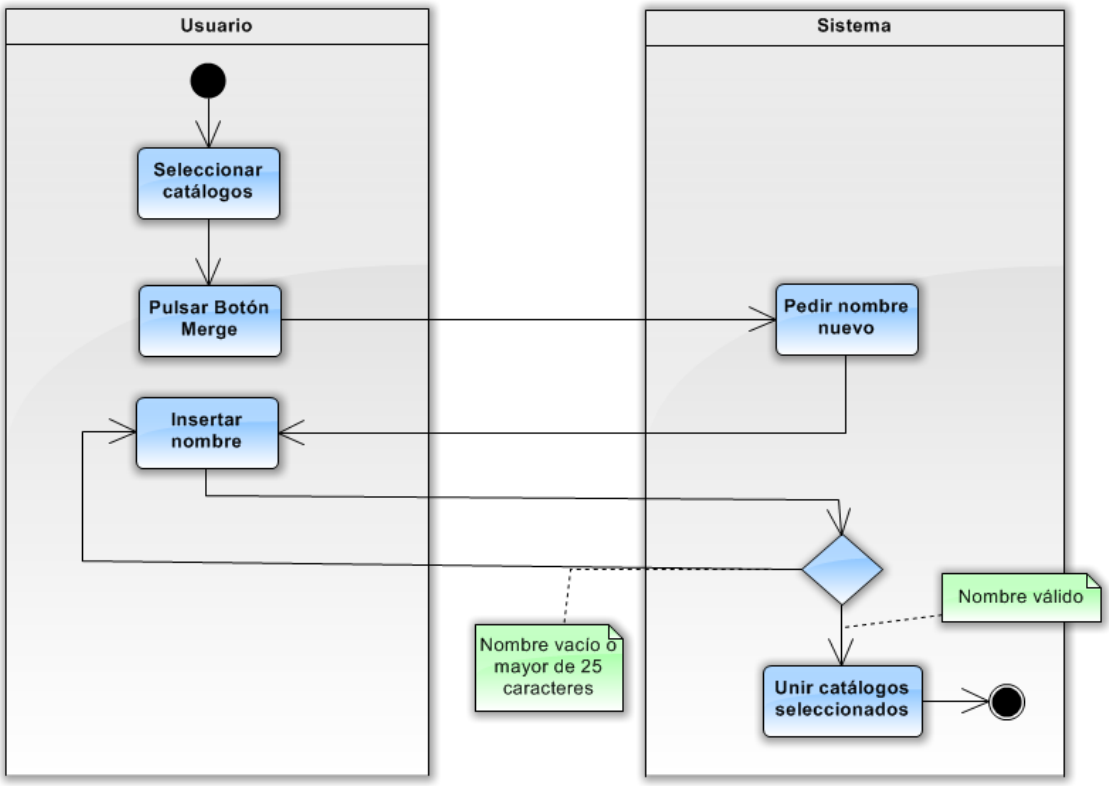


Figura 12 Diagrama de Actividad Merge

En este primer módulo las posibles complicaciones fueron detectadas inmediatamente, siendo la primera de ellas la importación de catálogos. Como se puede ver en la Especificación de Requisitos Software (apartado 1.3.1.1), los catálogos, pese a poder tener formatos diferentes (es decir, las estrellas de un catálogo tienen el mismo formato entre sí, pero pueden ser diferentes a las de otro catálogo), debían comenzar por una cabecera VizieR válida. De esta cabecera válida era necesario obtener toda la información necesaria, como los datos del catálogo o los campos de las estrellas, y además, leer una a una cada estrella para poder persistirlas en la base de datos de la aplicación. Se requería por tanto el uso de un parseador que se encargara de leer línea a línea el fichero de texto, diferenciando las partes de éste y procesando toda la información. Para tal tarea, se implementó dicho parseador como un autómata finito determinista.

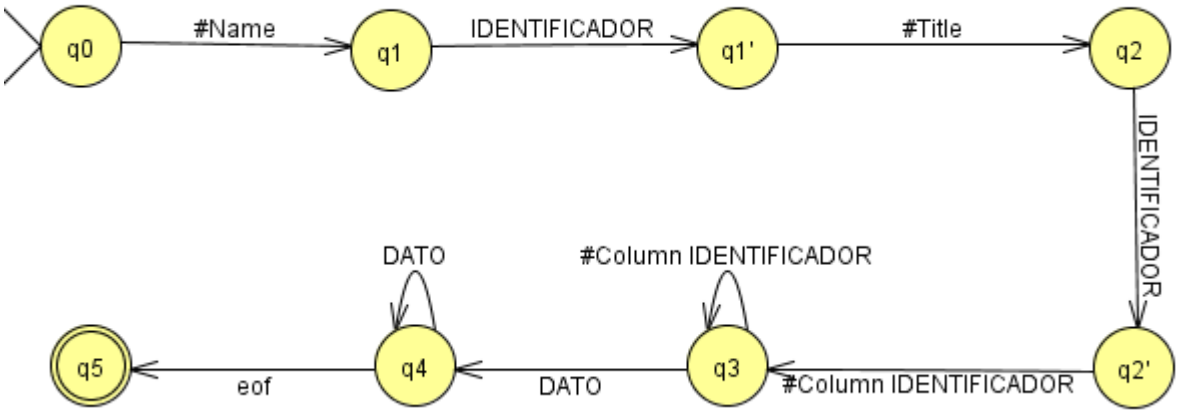


Figura 13 Automata Finito utilizado para parsear el fichero de entrada

Sin embargo, aunque el comportamiento de la aplicación en este aspecto era el esperado, el rendimiento dejaba mucho que desear. Esto suponía dos problemas, el primero, que obligaba a la aplicación a estar funcionando durante largos intervalos de tiempo (horas) para insertar un catálogo. El segundo, que durante este tiempo, la aplicación estaba bloqueada y no se podía realizar ninguna operación más. La solución al primero de estos percances se puede hallar de forma más detallada más adelante en este documento, pero básicamente se basaba en reducir en la medida de lo posible las llamadas a la base de datos. Aun así, nos parecía bastante razonable que el usuario pudiese hacer uso de las otras funcionalidades de la aplicación aun cuando se estuviese importando un catálogo, o incluso ofrecer la posibilidad de importar más de un catálogo al mismo tiempo. Para ello, recurrimos a la API de Java encargada de la gestión de hilos, es decir, queríamos que cada proceso de importación se convirtiese en un hilo diferenciado, y que el usuario tuviese a su disposición un panel que le informara de que catálogos se estaban importando y le diera la posibilidad de finalizar en cualquier momento cualquier importación, sin perder el trabajo realizado hasta el momento. A pesar de ciertos problemas de concurrencia en el acceso a las tablas, finalmente se pudo implementar esta funcionalidad de forma exitosa. Importar catálogos había dejado de ser un proceso largo y tedioso.

El siguiente problema se hallaba en la operación principal de este módulo, la que permitía cruzar catálogos de cualquier tipo entre sí con el fin de encontrar pares de estrellas con movimiento propio común. La idea de partida era conseguir una nueva tabla que englobara el producto cartesiano de dos catálogos, y a partir de esta macro tabla, realizar el filtrado con los parámetros de minería de datos que deseásemos. Sin embargo, esta aproximación era muy poco eficiente. Si consideramos que por ejemplo un catálogo puede tener normalmente 200.000 estrellas y quisiéramos cruzarlo consigo mismo, obtendríamos como resultado una macro tabla de 200.000x200.000 estrellas, es decir, 40.000.000.000 entradas. Estamos hablando por tanto de procedimientos que podían tardar días o incluso semanas en completarse. Además, como ya hemos comentado, las estrellas de dos catálogos podían ser completamente diferentes, o se podía dar el caso de que cruzásemos catálogos que contuvieran estrellas únicamente del hemisferio norte, con otro que solo contuviera estrellas del hemisferio sur.

Esta última situación que acabamos de mencionar nos dio una pista de por donde se podía atacar el problema. Nosotros buscábamos pares de estrellas que debían estar excepcionalmente cerca. Qué sentido tenía pues comparar estrellas de distintos hemisferios si era evidente que no cumplían los requisitos de cercanía mínimos. Por tanto, se llevó a cabo una nueva aproximación al problema en la cual, en vez de crear una macro tabla con el producto cartesiano de los dos catálogos, se iban creando una serie de minitablas que albergaban pares de estrellas dentro en un intervalo de cercanía razonable (por ejemplo, un grado), para a continuación, realizar sobre ellas el filtrado. Con esta técnica, no solo reducíamos de forma drástica el número de operaciones que tenía que realizar el programa, sino que además, en el ejemplo anterior de los dos hemisferios, al no haber un solo par de estrellas lo suficientemente cerca, no se creaba ninguna minitabla y la aplicación ofrecía el resultado de la operación inmediatamente. En el siguiente diagrama de secuencia, podemos observar la llamada al método que se encarga de realizar todo este proceso algorítmico junto con el resto de las llamadas internas que se generan en este caso de uso.

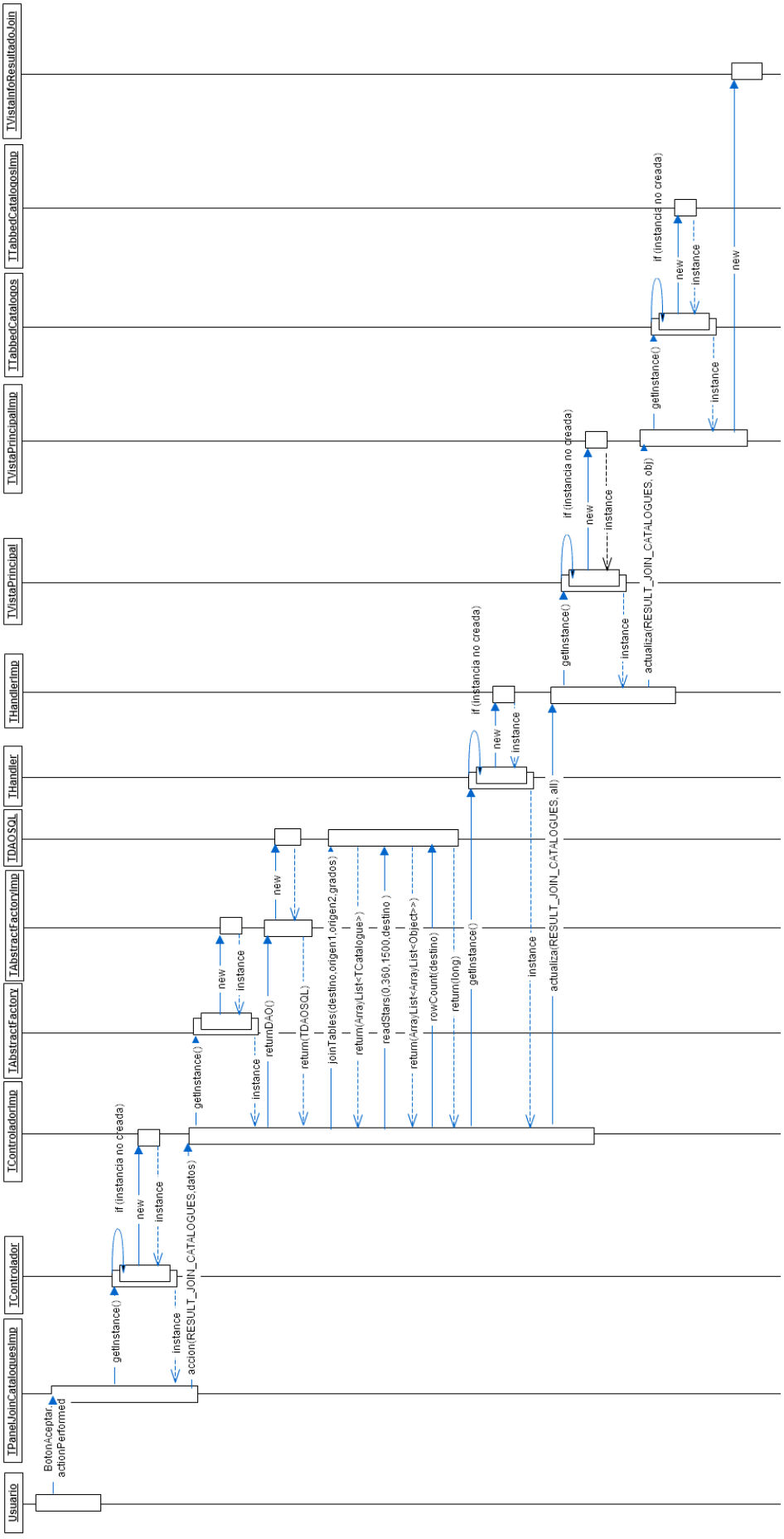


Figura 14 Diagrama de Secuencias de Join Catalogues

Para realizar estas comparaciones por cercanía también nos surgió una pequeña disyuntiva: ¿qué campos se iban a comparar?

Como ya hemos comentado, cada estrella podía estar codificada con un formato distinto, pero al menos debían tener unas coordenadas. No tenía sentido trabajar con unas estrellas de las cuales no supieras su localización en el orbe celeste. Sin embargo, estas coordenadas podían haber sido denominadas de cualquier manera en la cabecera del catálogo, podían ser simplemente RA y DE, o RAJ2000 y DEJ2000, o usar guiones bajos…

Para hallar la solución a este nuevo percance, empezamos a analizar todos los catálogos que habíamos descargado previamente para descubrir que VizieR, automáticamente, creaba dichos campos al descargar el catálogo siempre con la notación _RAJ2000 y _DEJ2000.

El problema a priori parecía solucionado, sin embargo, esto no nos garantizaba que en un futuro VizieR no pudiera cambiar la notación con la que se descargaban los catálogos. Es en este punto donde el grupo de desarrollo, a petición del director del proyecto, decide incluir un nuevo requisito en la aplicación, la creación de cabeceras. A través de esta nueva funcionalidad, ahora no dependeríamos de la notación usada en VizieR. El usuario tendría un menú con el cual podría crear su propia cabecera para cualquier fichero de texto que contuviera estrellas, definir sus campos (que por supuesto debían encajar con el contenido de dicho fichero), y a continuación importar el catálogo. Con esto conseguimos mejorar la funcionalidad de la aplicación de forma notable. Por un lado, CPMP procesaría cualquier tipo de catálogo independientemente de si tuviera cabecera o no, por otro, nos ahorrábamos la dependencia de VizieR, y si en un futuro la comunidad científica decidía cambiar de notación, el usuario podría seguir usando la aplicación creándose sus propias cabeceras para los nuevos catálogos.

Otra de las funcionalidades que podíamos considerar complicada de implementar era la de visualizar por pantalla placas fotográficas de las estrellas a través de Aladin. Aladin, como quedó contemplado en la sección 1.2.1 de la Especificación de Requisitos Software , es otro de los servicios online del Centre de Données Astronomiques de Strasbourg, y su finalidad es recoger y tener accesibles

para los usuarios vía Internet placas fotográficas de estrellas. El CDS proporciona a los usuarios bastante documentación para manejar sus servicios. En dicha documentación se manejaban dos opciones, o bien usar el applet web de la página de Aladin, o intentar integrar el Jar con las API´s correspondientes dentro de nuestra aplicación.

En un primer momento nos inclinamos por esta última opción, pues la considerábamos más flexible y no requería de elementos externos como navegadores web, sin embargo, tras varios días peleándonos con la API para intentar cargar imágenes a partir de las coordenadas de una estrella, tuvimos que desistir de la tarea. Era fácil usar el Jar desde la consola de comandos, pero intentando realizar un proceso similar a partir de código Java tal y como se indicaba la documentación, conseguíamos hacer uso de todas las funcionalidades del API menos la que nos interesaba, la de cargar imágenes a partir de coordenadas. Tras una reunión con el director del proyecto, este nos informó de que la metodología que estábamos usando no era válida, y que a pesar de que según la documentación nuestro código era correcto, para alcanzar el resultado se necesitaba el desarrollo de una serie de scripts extremadamente complejos. No sin mucho pesar y tras algún par de intentos más, descartamos pues la posibilidad de integrar el Jar, y decidimos enfocar la implementación por la vía del applet web. Aquí nuevamente apareció una nueva disyuntiva, teníamos la posibilidad de usar un navegador externo, o de intentar renderizar la página web con el motor de Java. Sabíamos que esta plataforma era capaz de cargar páginas web en algunos de sus componentes gráficos como los JPanel, por lo que nuevamente elegimos la opción que no requería de elementos externos. El primer paso necesario para usar este applet era generar las URL´s, es decir, a partir de las coordenadas de las estrellas, construir una dirección web que accediera al applet de la página web de Aladin para que este mostrara las estrellas adecuadas.

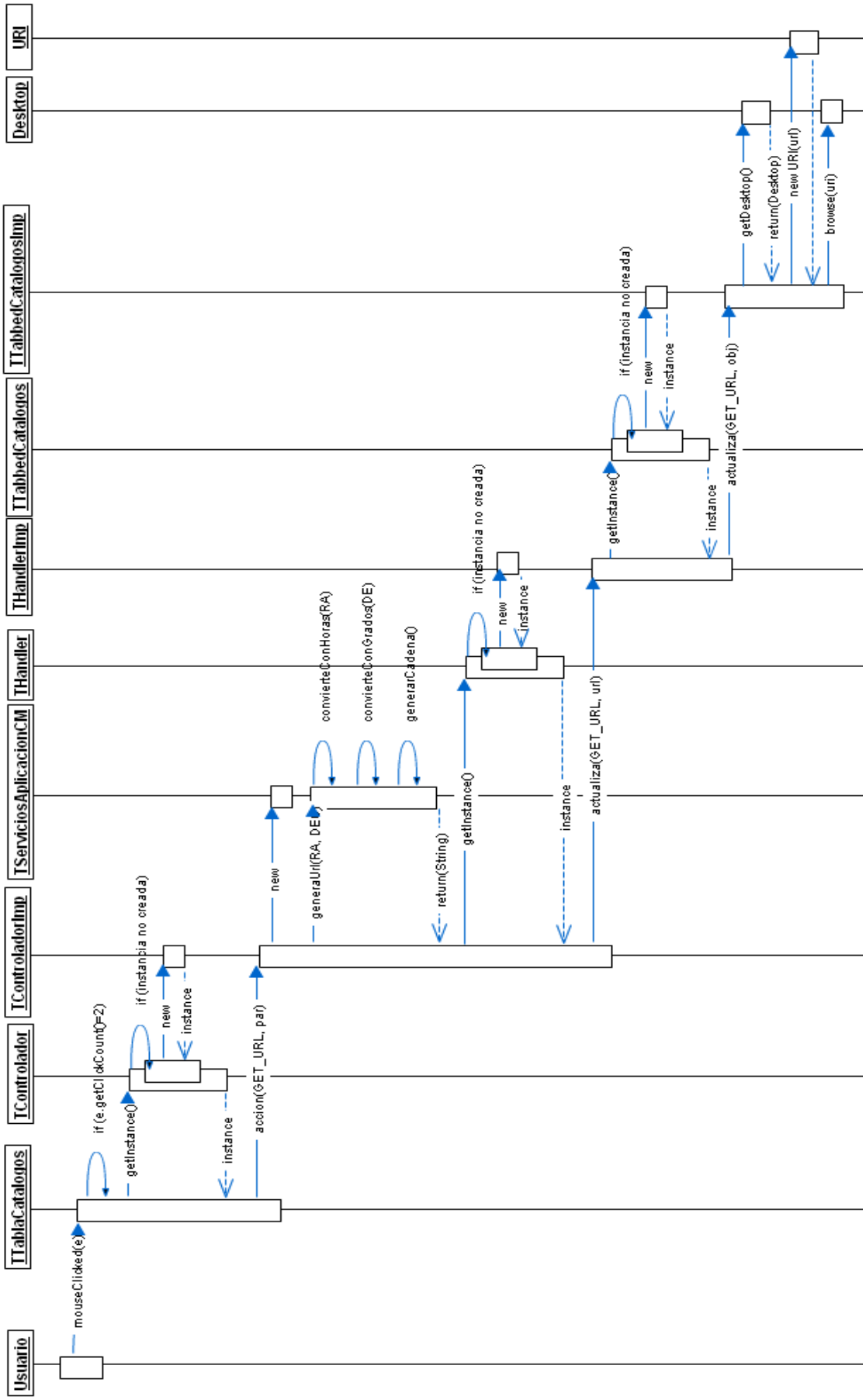


Figura 15 Diagrama de Secuencias de Mostrar estrella

Como podemos ver en la anterior figura este proceso se dividió en dos partes, la primera, transformar las coordenadas de grados decimales a grados sexagesimales (requeridos por Aladin), y la segunda, generar la cadena de texto adecuada. Aladin disponía de una funcionalidad, que a partir de un pequeño script generaba la URL equivalente, por lo que simplemente escribimos el script que nos interesaba y analizamos la URL generada por Aladin para descubrir los patrones utilizados en su construcción. Sin embargo, nuevamente nos llevamos una decepción, ya que el motor de Java no era suficientemente potente. Tras una nueva labor de investigación, constatamos que Java únicamente permitía renderizar código html muy sencillo, por lo que desde luego, era imposible cargar un applet dentro de un JPanel. Por tanto, acabamos recurriendo a la última de las opciones, es decir, cargar las imágenes en un navegador web externo a la aplicación, y aunque el resultado no era tan elegante como lo que habíamos intentado conseguir hasta el momento, la simple posibilidad de poder cargar imágenes de las estrellas con un simple doble click en nuestra aplicación merecía la pena. Ya que el proceso de generar URL´s a partir de coordenadas ya había sido desarrollado, simplemente hicimos uso de las llamadas al sistema adecuadas para abrir un navegador web con la URL que nosotros proporcionábamos.

Para finalizar con este módulo, deseábamos que se pudieran exportar los catálogos ya incluidos en la aplicación a ficheros de texto. Supongamos que un usuario ha cruzado varios catálogos entre sí, y tiene como resultado un nuevo catálogo con 200 pares de estrellas candidatas a ser “dobles”. Es razonable pensar que el usuario quiera poder transportar esa información a otros ordenadores o imprimirla en papel, por lo que haciendo uso de funcionalidades del motor de bases de datos SQL que estábamos usando, se implementó la posibilidad de poder convertir cualquier catálogo presente en la aplicación en un fichero de texto con una cabecera válida.

El segundo módulo de la aplicación, el que tenía como finalidad permitir al usuario el desarrollo de scripts, no tenía en principio tantas funcionalidades como el de gestión de catálogos, sin embargo, estas iban a requerir de más reflexión y trabajo que las anteriores. Nuevamente, empezamos por definir unas ideas básicas a partir de la especificación de requisitos obtenida con el director de proyecto.

Queríamos un panel en el cual se pudiese escribir y ejecutar un script (para mayor información sobre cómo crear un script válido, ir a la sección 1.3.1.2 de la Especificación de Requisitos Software), un listado de catálogos importados a la aplicación, y un botón para ejecutar las instrucciones del script. Para realizar esta tarea, la aplicación debía ser capaz de reconocer y llevar a cabo un conjunto básico de instrucciones, lo que suponía aplicar técnicas de procesamiento de lenguajes. Esta fase estaba compuesta inicialmente de dos etapas, por un lado la compilación de los scripts y la generación del código objeto correspondiente y, por otro, la ejecución de dicho código. Para entender el trabajo que debía ser llevado a cabo es necesario exponer una serie de conceptos previos:

Analizador sintáctico

Fase del analizador que se encarga de chequear el texto de entrada en base a una gramática dada. En caso de que el programa de entrada sea válido, suministra el árbol sintáctico que lo reconoce. En teoría, se supone que la salida del analizador sintáctico es alguna representación del árbol sintáctico que reconoce la secuencia de tokens suministrada por el analizador léxico.

El analizador sintáctico acepta gramáticas incontextuales o gramáticas independientes del contexto (GIC). Las GIC permiten la existencia de recursividad en los lenguajes que representan, así como la aparición de estructuras anidadas (por ejemplo: estructuras if-then anidadas, paréntesis anidados en expresiones aritméticas, etc).

Dicha recursividad implica:

- ◊ Algoritmos de reconocimiento más complejos, que hagan uso de llamadas recursivas o usen explícitamente una pila, la *pila de análisis sintáctico*.
- ◊ La estructura de datos usada para representar la sintaxis del lenguaje ha de ser también recursiva (el árbol de análisis sintáctico).

El formato de una GIC es el siguiente:

Gramática: G (**N**, **T**, **P**, **S**)

N = No terminales.

T = Terminales.

P = Reglas de Producción.

S = Axioma Inicial.

Derivación: Se considera una producción como una regla de reescritura, donde el no terminal de la izquierda es sustituido por la cadena del lado derecho de la producción.

Derivación por la izquierda: Derivación donde solo el no terminal de más a la izquierda de cualquier forma de frase se sustituye en cada paso.

Derivación por la derecha: Derivación donde el no terminal más a la derecha se sustituye en cada paso.

Árbol sintáctico de una sentencia de un lenguaje: Representación utilizada para describir el proceso de derivación de dicha sentencia. Como nodos internos del árbol se sitúan los elementos no terminales de las reglas de producción que vayamos aplicando y tantos hijos como símbolos existan en la parte derecha de dichas reglas. Si el árbol puede construirse, es que la sentencia es correcta.

Análisis ascendente: Se parte de la cadena de entrada para construir la inversa de una derivación por la derecha. Genera el árbol de análisis sintáctico partiendo de las hojas hasta alcanzar el axioma.

Ventajas

- Es el método más genérico que se conoce y es tan eficiente como los demás.
- Reconoce más gramáticas que los analizadores predictivos.
- Reconoce muchos lenguajes independientes del contexto.

Inconvenientes

- Diseño muy costoso para lenguajes de programación.
- Realiza muchas comprobaciones de la cadena de entrada.

Análisis descendente: En cada paso del proceso de derivación de la cadena de entrada se realiza una predicción de la posible producción a aplicar y se comprueba si existe una concordancia entre el símbolo actual en la entrada con el primer terminal que se puede generar a partir de esa regla de producción. Si existe

esta concordancia se avanza en la entrada y en el árbol de derivación, en caso contrario se vuelve hacia atrás y se elige una nueva regla de derivación.

Ventajas

- Son los más potentes.

Inconvenientes

- Son más lentos.
- Aceptan gramáticas ambiguas.
- La recursividad a izquierdas da lugar a un bucle infinito de recursión.
- Problemas de indeterminismo cuando varias alternativas en una misma producción comparten el mismo prefijo (factorización).

Una vez definidos estos conceptos, se decidió implementar un analizador descendente predictivo recursivo, ya que su diseño es más sencillo y fácil de mantener. Dicha elección provoca que la gramática representativa del lenguaje sufra un proceso de transformación en el que se deben eliminar la recursión izquierda y la factorización. Las reglas de dichas transformaciones son las siguientes:

En nuestro caso sólo fue necesario aplicar la primera regla, ya que la gramática de origen no contenía producciones en las que fuera necesaria la factorización. A continuación se muestran ambas gramáticas:

GRAMÁTICA INCONTEXTUAL DEL LENGUAJE

Prog --> Begin Sents End

Sents --> Sents SAsig| SAsig

SAsig --> IDENTIFICADOR <- Exp ; [i]

Exp --> IDENTIFICADOR (Params) Filtro

Params --> Params, Param | Param

Param --> IDENTIFICADOR | 'EXPRESION' [ii]

Filtro --> [Filt] | λ

Filt --> CONDICION SQL | NUMERO

[i] Con esta producción obligamos a que toda instrucción acabe con el símbolo Terminal ";"

[ii] Una expresión numérica o booleanas, según el caso

GRAMÁTICA INCONTEXTUAL TRANSFORMADA

Prog --> Begin Sents End

Sents --> SAsig RSents

RSents --> SAsig RSents

RSents --> λ

SAsig --> IDENTIFICADOR <- Exp ; [i]

Exp --> IDENTIFICADOR (Params) Filtro

Params --> Param RParams

RParams --> , Param RParams

RParams --> λ

Param --> IDENTIFICADOR | 'EXPRESION' [ii]

Filtro --> [Filt] | λ

Filt --> CONDICION SQL | NUMERO

[i] Con esta producción obligamos a que toda instrucción acabe con el símbolo Terminal "**;**"

[ii] Una expresión numérica o booleanas, según el caso

Para entender mejor como se realiza el proceso de derivación de una entrada valida en nuestro lenguaje mostramos un árbol de derivación de un ejemplo concreto de script. Para ello se han seguido las producciones descritas en las gramáticas anteriores.

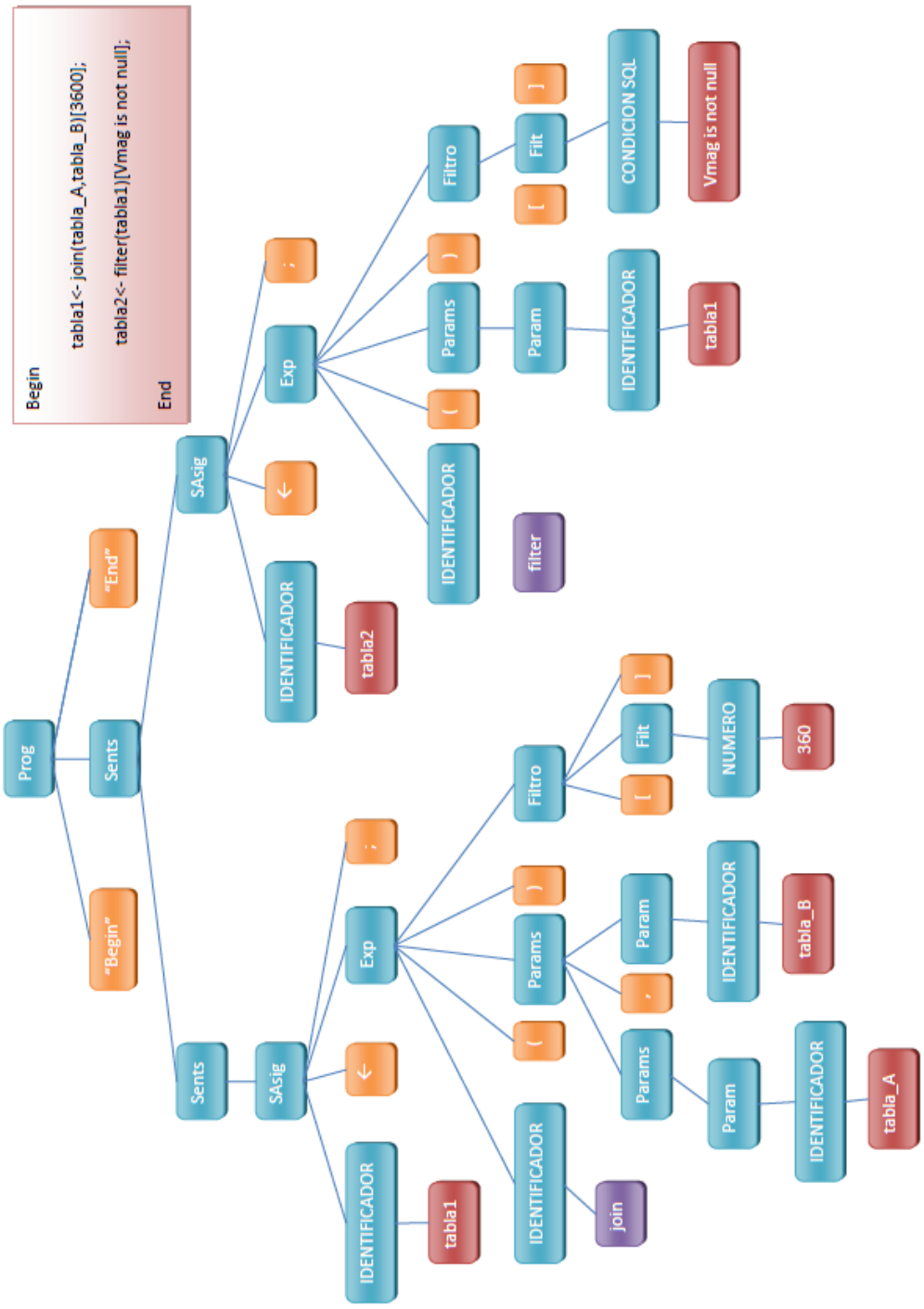


Figura 16 Árbol derivativo de la Grámatica Incontextual

Llegados a este punto nos dimos cuenta de que la complejidad de los scripts no era suficiente como para diseñar un compilador. Un compilador lee totalmente un programa y lo traduce a otro programa equivalente en un lenguaje más sencillo, el código objeto. Posteriormente este código objeto es ejecutado en su totalidad. Esto tiene una serie de ventajas: el aumento del rendimiento y la posibilidad de portar el código objeto a otras máquinas. Sin embargo ninguna de estas cualidades se ajustaba a lo que nuestra funcionalidad demandaba. Por otro lado, existía la opción de implementar un intérprete que, a diferencia de un compilador, no produce código objeto sino que por cada sentencia que compone el texto de entrada, se realiza una traducción, ejecuta dicha sentencia y vuelve a iniciar el proceso con la sentencia siguiente. Los intérpretes suelen ser usados para ejecutar lenguajes de órdenes, pues cada operador que se ejecuta en un lenguaje de este tipo es una invocación a una subrutina.

En nuestro caso las instrucciones que conforman un script no podían ser descompuestas en otras más sencillas, sino que equivalían a una subrutina SQL, además el rendimiento no iba a ser un factor determinante ya que el código de los scripts no entrañaba excesiva complejidad. Sin lugar a dudas, la implementación de un intérprete se ajustaba perfectamente a nuestras necesidades.

Una vez identificada cada operación y desglosados sus parámetros, la instrucción era ejecutada y su resultado mostrado en la consola. En la siguiente figura encontrarán el diagrama de secuencia de este proceso.

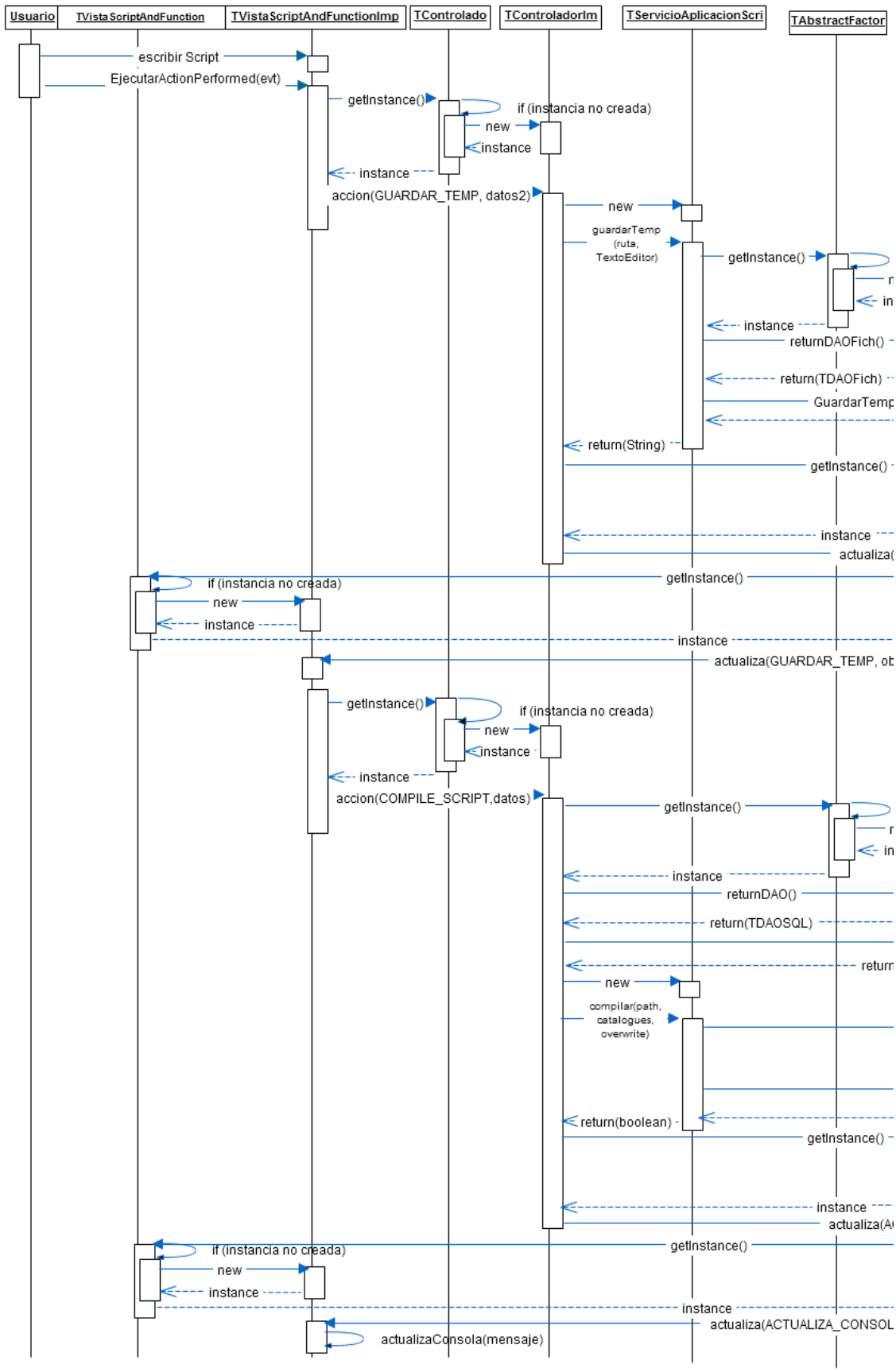


Figura 17.A Diagrama de Secuencias de Script

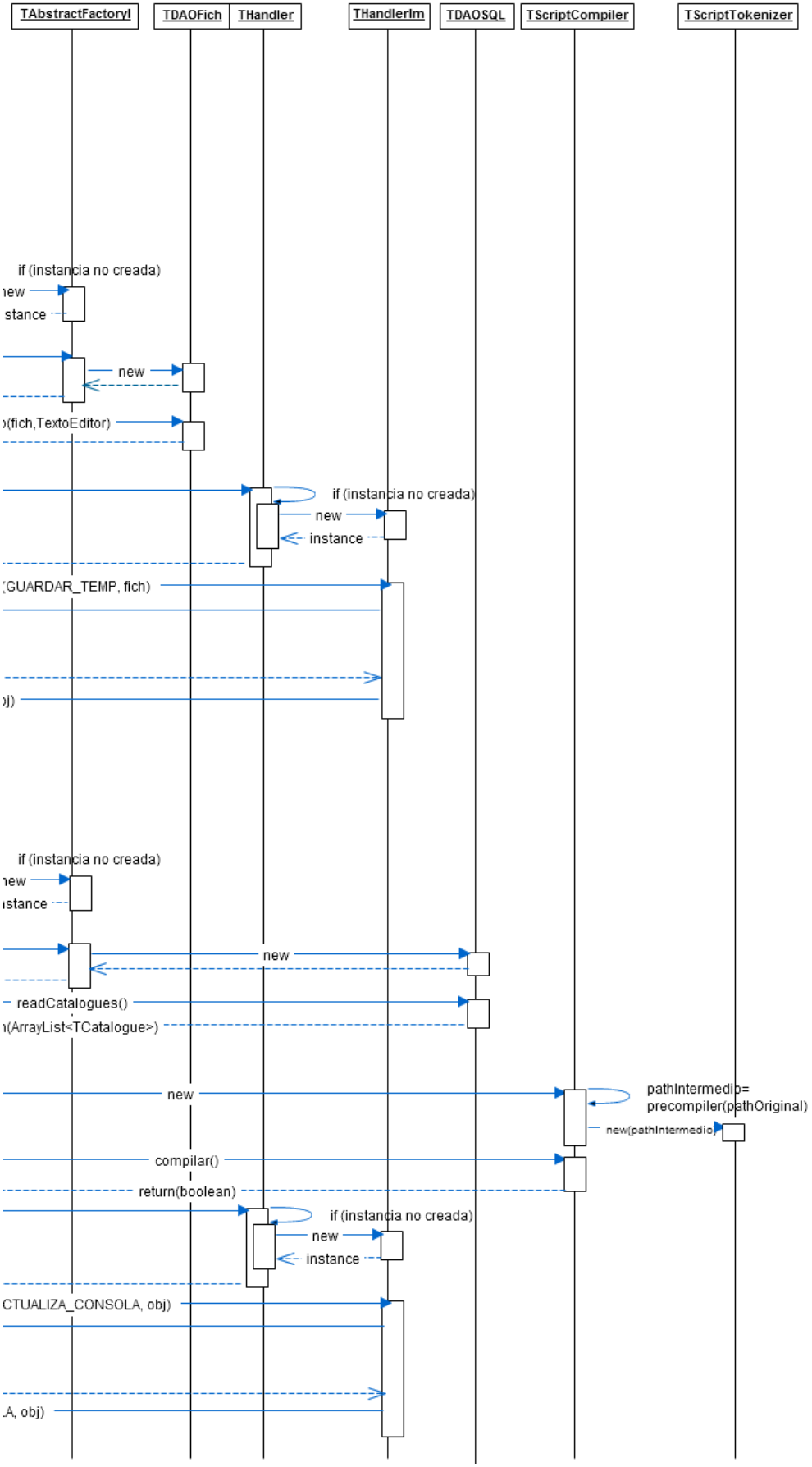


Figura 17.B Diagrama de Secuencias de Script

A la hora de trasladar la segunda gramática a Java, nos fueron de gran utilidad las API´s de Java StringTokenizer y StreamTokenizer. Ambas tienen una funcionalidad básica, coger una línea de texto y devolverla en partes (tokens), pudiendo seleccionar el programador qué caracteres harán de separadores. La diferencia entre ambas API´s es que StringTokenizer es algo más simple. Por ejemplo, no diferencia números o identificadores, ni reconoce o salta comentarios.

Este último API fue utilizado para detectar las funciones que habían sido usadas en el script. Como ya indicábamos de forma mucho más detallada en la Especificación de Requisitos Software, el usuario tiene la opción de crearse sus propias "librerías" o "funciones" a partir de las operaciones atómicas que tiene a su disposición. En este caso, necesitábamos una herramienta que simplemente nos detectara que se estaba usando una función, y no una de las operaciones básicas. En este caso, se debía buscar esta función o librería dentro del disco del usuario, e incrustar el nuevo código sustituyendo los parámetros.

Para distribuir de forma apropiada todas estas funcionalidades y ofrecer una interfaz de servicios unificada, se tomó la decisión de utilizar el patrón "Servicio de Aplicación", encargado de centralizar y encapsular en la medida de lo posible toda la lógica de la aplicación.

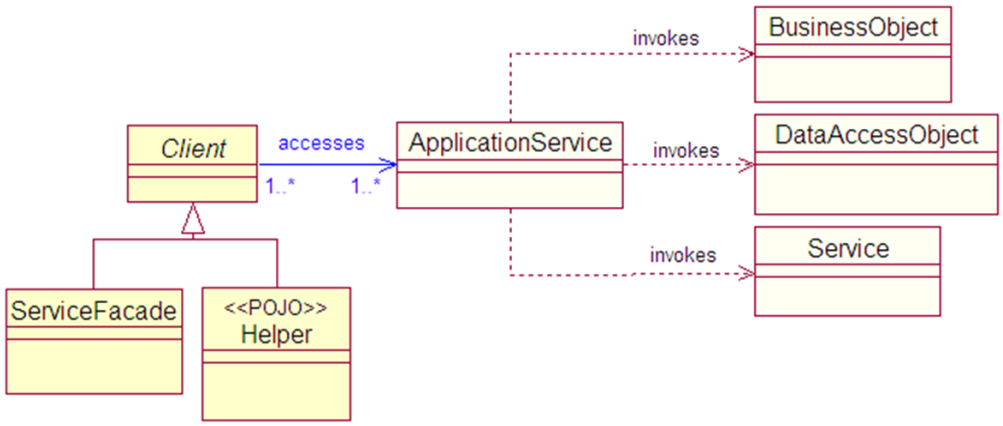


Figura 18 Estructura del Patrón Servicio de Aplicación

En el caso del módulo de desarollo de scripts por ejemplo, dicho Servicio de Aplicación recibe la orden de interpretar un script. Para ello, recurre a elementos externos como una clase "TCompilador" que realiza todas las tareas comentadas

anteriormente, para a continuación, ejecutar él mismo cada una de las instrucciones. En el caso del módulo de gestión de catálogos, el servicio de aplicaciones recibe todas las demandas de servicios, desde la de parsear el fichero de texto, hasta la de crear la URL apropiada para acceder a Aladin y visualizar una estrella.

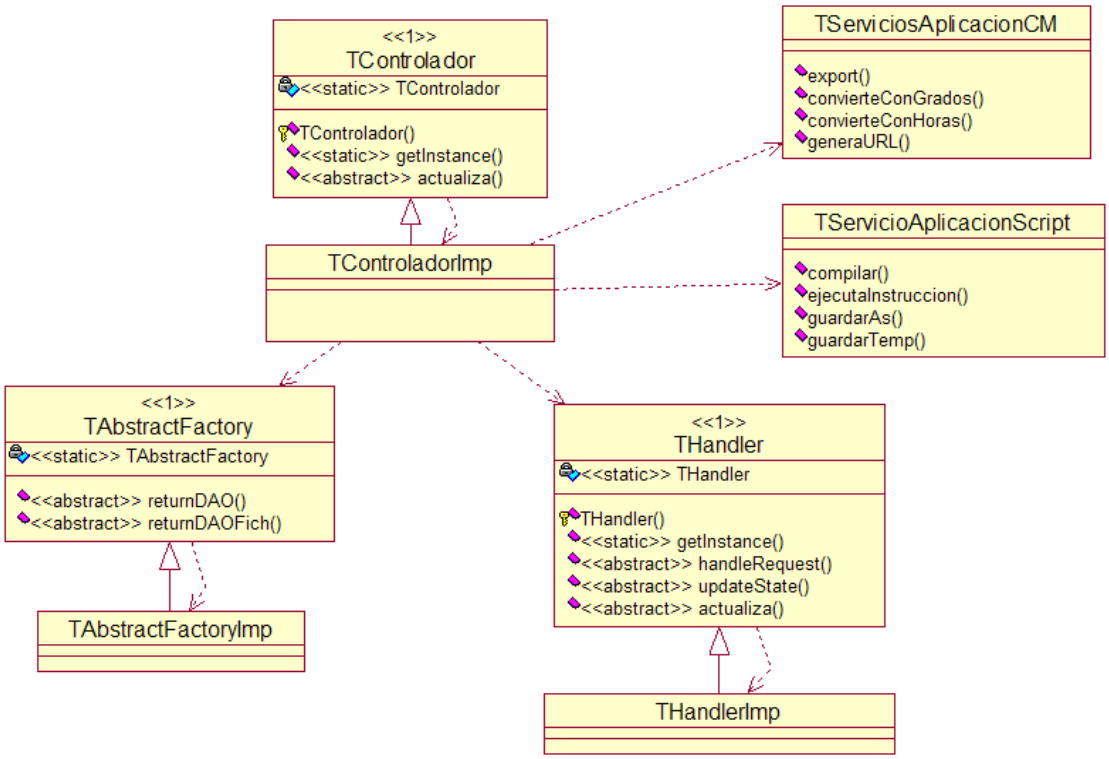


Figura 19 Diagrama de clases del Servicio de Aplicación

A lo largo de esta sección, nos hemos referido en varias ocasiones al uso de motores de persistencia SQL, y aunque entraremos en más detalles en el siguiente apartado, es conveniente introducir ahora el concepto de Factoría Abstracta. El patrón Factoría Abstracta proporciona una interfaz para crear familias de objetos relacionados o que dependen entre sí, sin especificar sus clases concretas. Desliga la creación de nuevos objetos de la clase que se refiere a dichos objetos a través de la interfaz que implementan.

A pesar de que podíamos haber usado Factorías Abstractas en otras partes del código, el caso del acceso a la capa de persistencia nos pareció especialmente intuitivo y apropiado para el perfil de la aplicación.

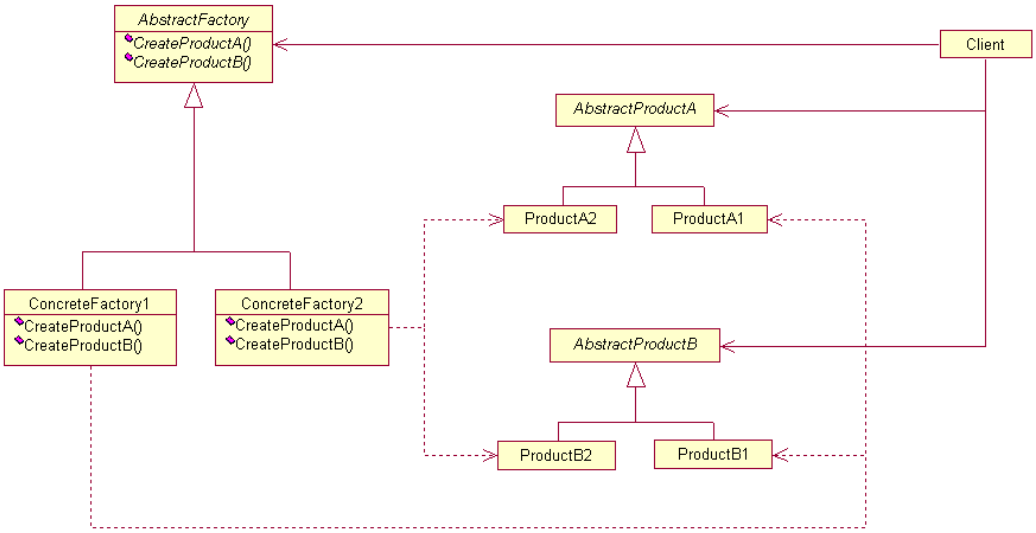


Figura 20 Patrón Factoría Abstracta

Como ya dijimos en la sección 1.2.6 de la especificación de requisitos software, en un futuro esta aplicación podría ser ampliada para usar otros motores de persistencia, ya sean bases de datos relacionales como Oracle o ficheros XML. Sin embargo, es evidente que el hecho de cambiar la forma en la que van a persistir los datos en la aplicación debería tener el mínimo número de ramificaciones posibles en el resto del código. El patrón Factoría Abstracta cumplía los requisitos que buscábamos exactamente.

El primer paso era crear una interfaz que agrupara las funcionalidades que la aplicación iba a demandar de la capa de persistencia, ya fuese esta implementada en SQL o XML, y eran estas interfaces, devueltas por la Factoría Abstracta, lo que estaba visible al resto de la aplicación, que no se veía afectada en ningún momento por las distintas implementaciones de dicha interfaz.

La traza de una operación, como por ejemplo, exportar un catálogo, sería la siguiente: el controlador, en la capa de presentación, ha capturado el evento y la ruta del fichero a la que exportar el catálogo. A continuación, demanda la ejecución de esta funcionalidad al Servicio de Aplicaciones del módulo de gestión de catálogos. Este Servicio de Aplicaciones, tras una serie de pre procesamientos, pide a la Factoría Abstracta “alguien” que sea capaz de exportar un catálogo. La Factoría

Abstracta devuelve un interfaz, por lo que el Servicio de Aplicaciones no sabe si los datos que se van a exportar han sido persistidos con MySQL o con XML. A continuación, el Servicio de Aplicación desea crear la cabecera, y nuevamente, tras una serie de pre procesamientos, pide a la Factoría Abstracta "alguien" capaz de introducir una cabecera válida en comienzo del fichero recién creado. Otra vez, el Servicio de Aplicaciones no sabe si el fichero generado es de texto, binario o tiene cualquier otro formato

3.3.3. Capa de persistencia

Como ya hemos dicho, esta capa es responsable de que la tecnología elegida para persistir la información permanezca opaca al resto del sistema. Su misión por tanto es mediar en el intercambio de datos entre las capas superior (negocio) e inferior (datos, a la cual nos referíamos como la "5ª capa" en la introducción de la arquitectura multicapa) mediante el uso de interfaces. Estará formada por los Data Access Objects, o DAOs, que son los únicos elementos en el sistema que conocen la representación de los datos dentro de los motores de persistencia. Los DAOs son creados por una Factoría Abstracta bajo demanda de la capa de lógica.

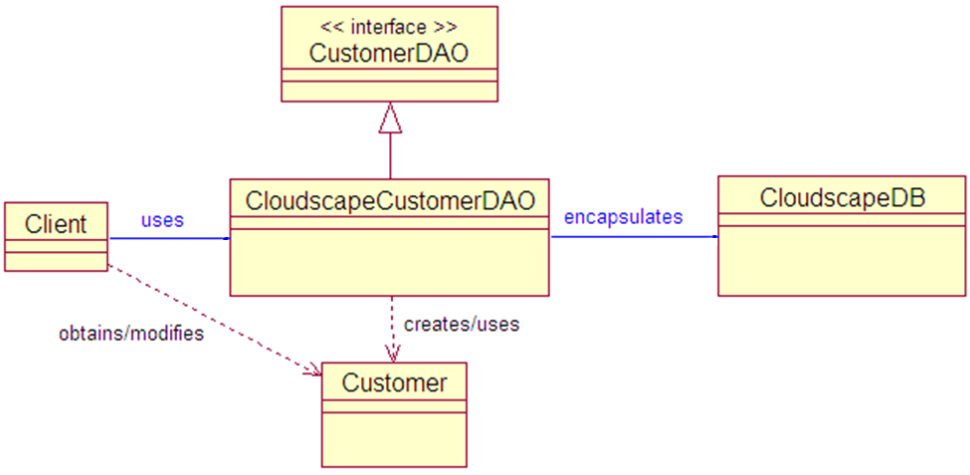


Figura 21 Patrón DAO

Nuestra capa de persistencia tendrá dos elementos principales, **TDAOSQL** y **TDAOFich** junto con sus correspondientes interfaces, encargados de ofrecer un contrato de servicios al resto de la aplicación.

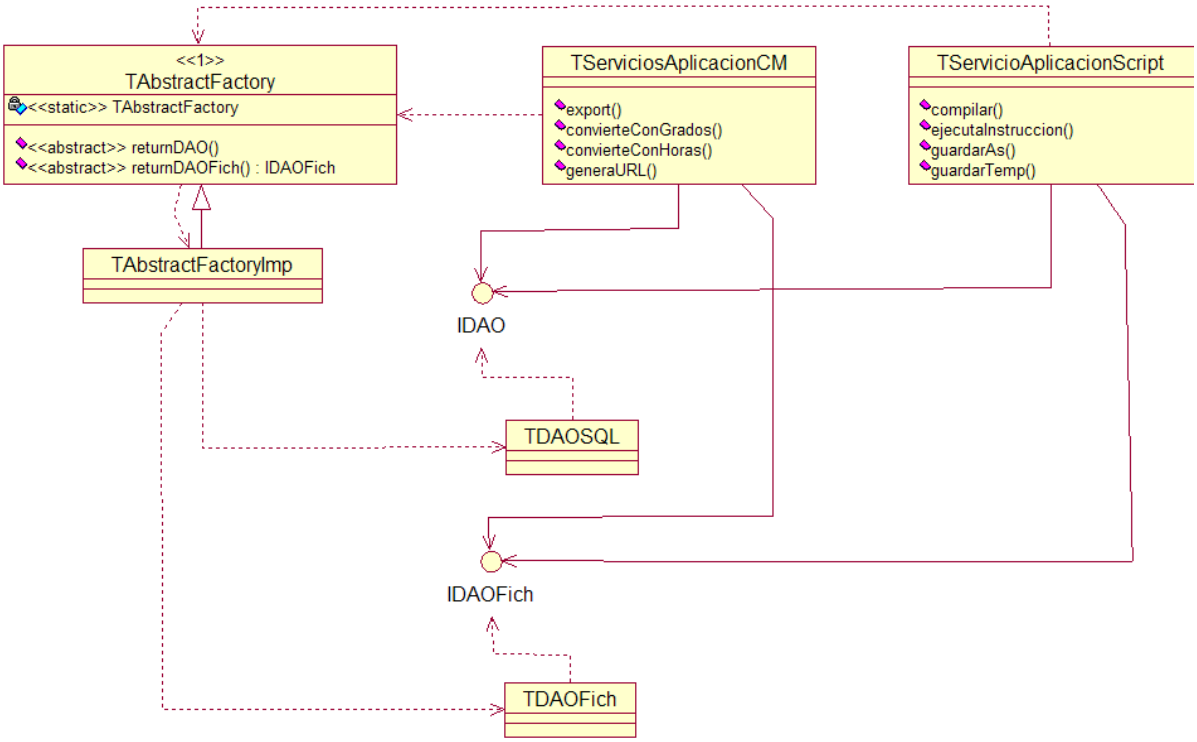


Figura 22 Diagrama de Clases de la capa de Persistencia

El primero de ellos, se encarga de interactuar con el motor de bases de datos relacional MySQL. Este DAO accede a dicha base de datos MySQL (previamente instalada en la máquina) mediante un objeto JDBC (Java DataBase Connector) que no es liberado hasta el fin de la aplicación. Para realizar el intercambio de datos, se han usado los objetos habituales Statement y ResultSet. Sin embargo, **TDAOSQL** no es únicamente una clase encargada de llevar a cabo transacciones, pues contiene también bastante algoritmia que era necesaria implementar en SQL por motivos de eficiencia y rendimiento. Es el caso por ejemplo de la opción de cruzar tablas, desarrollada en mayor profundidad durante la sección anterior, que está implementada prácticamente en su totalidad con instrucciones SQL o la operación MINUS del script, que permite "restar" un catálogo a otro.

Para conseguir aumentar el rendimiento de la aplicación lo máximo posible en las operaciones que requerían una mayor carga del sistema, como la importación de catálogos a partir de ficheros de texto, se han utilizado algunas técnicas transaccionales o propias del funcionamiento interno de las bases de datos. Por ejemplo, en dicho caso de importación de catálogos, el objeto **TDAOSQL** recibía un array polimórfico que encapsulaba toda la información concerniente a una nueva estrella. A partir de estos datos, generaba la instrucción SQL correcta e insertaba la estrella dentro de la base de datos. De esta manera, en un catálogo con 200.000 estrellas, se acababan ejecutando un número al menos igual de llamadas a la base de datos. Para optimizar este proceso se tomó la decisión de usar buffers donde se iban cargando todas las estrellas posibles (teniendo siempre precaución con el consumo de memoria), para finalmente hacer un "commit". De esta forma, si con cada llamada a la base de datos insertábamos 1500 estrellas, el número de estas llamadas se reducía drásticamente aumentando notablemente el rendimiento de la aplicación. El mayor problema que nos encontramos durante la implementación de esta capa, más allá de lo ya comentado, fue un problema con los tipos de datos que consiguió exasperarnos. Para explicarlo de forma resumida, el tipo "float" de Java no tenía la misma precisión en número de decimales que el tipo "float" de SQL. Esto significaba que si por ejemplo nosotros queríamos insertar una estrella con los seis decimales de precisión, en MySQL solo se guardaban 5 de estos decimales. Este en un principio no parece un problema crítico, sin embargo, suponía que la opción de navegar por los catálogos no funcionara correctamente. Si el usuario deseaba visualizar el siguiente intervalo de estrellas, internamente se generaba un nuevo intervalo utilizando como coordenada de partida el final del anterior intervalo incrementado lo mínimamente posible, es decir, solo el último decimal, el que no llegaba a MySQL, era modificado. Esto significaba, que el punto de partida del nuevo intervalo incluía siempre la última estrella del intervalo anterior, y dado que en un principio la longitud de los intervalos estaba predefinida, en el momento en que dentro de un intervalo había una sola estrella, aunque el usuario quisiera avanzar, siempre se acababa mostrando la misma información. Hasta que conseguimos detectar donde estaba el problema pasaron

días. A raíz de esta situación, se implementó la posibilidad de que el usuario pudiera elegir la longitud del intervalo de estrellas que quería mostrar.

Por otra parte, **TDAOFich** será el encargado de trabajar con los ficheros de texto. Tras esta clase no subyace un concepto tan definido como en **TDAOSQL**, sin embargo decidimos utilizar un patrón DAO también para el acceso a ficheros de textos por motivos de coherencia. **TDAOFich** tendrá entre otras labores escribir cabeceras o realizar copias de ficheros cuando sea necesario.

3.4. GESTIÓN DEL RIESGO

1. Descomposición del grupo

El proyecto está pensado para que sea realizado por un número exacto de 3 personas. El fallo de alguno de los miembros puede ser desastroso para la consecución final del mismo.

Uno de los integrantes del equipo de desarrollo trasladó a vivir a otro país. Sin embargo, esta decisión fue tomada cuando el proyecto ya había sido completado, por lo que no ha tenido impacto en el resultado final y no se puede hablar de descomposición de grupo.

2. Posibilidad de bloqueo

A la hora de desarrollar el proyecto es necesario realizar labores de investigación que conduzcan a alcanzar los diversos objetivos marcados. Una situación de bloqueo o un resultado insatisfactorio en dicha investigación podrán retrasar o imposibilitar la consecución de dichos objetivos.

Sufrimos un bloqueo importante cuando no fuimos capaces de integrar el plug-in de Aladin en nuestra aplicación Java. Sin embargo, sí se habían definido previamente vías alternativas de conseguir resultados similares, por lo que se consiguió solventar la situación.

3. Planificación poco realista o poco equilibrada

Nuestra poca o nula experiencia en el desarrollo de este tipo de sistemas hace que podamos tener problemas al estimar el esfuerzo necesario y por ende, la planificación o realizar una división de la carga de trabajo poco equilibrada.

Podemos afirmar categóricamente, que nuestra planificación ha sido un éxito, y se ha seguido prácticamente de forma rigurosa, algo que en parte debemos agradecer al director del proyecto por mantenerse siempre accesible y dispuesto a llevar una carga de trabajo aceptable respecto a nuestro proyecto.

4. Poca experiencia con las herramientas

Para la realización del proyecto manejaremos herramientas como los programas Eclipse, Tortoise, servidores SVN, MySQL, VizieR, Aladin,..., algunos de ellos novedosos en nuestra formación. Una mala asimilación de su entorno y funcionalidad puede entorpecer sobremanera los avances y desajustar toda la planificación.

Salvo el caso ya comentado de Aladin, no hemos encontrado problemas en el uso de estas herramientas.

5. Desajuste en la planificación

Es bastante probable que, por algún motivo impredecible, nos sea imposible cumplir uno o más de los plazos marcados, lo que afectará al resto de la planificación.

Como ya hemos comentado anteriormente, la planificación se ha cumplido estrictamente, y pese a que hemos sufrido algún contratiempo inesperado, o alguno de los miembros del grupo no ha estado disponible de forma temporal, hemos sabido cumplir siempre los plazos marcados.

6. Problema con los laboratorios

Algunas de las herramientas que vamos a utilizar son de pago, por lo que solo podremos acceder a ellas mediante los equipos de la facultad. Además, en general, el acceso al software que necesitaremos para el desarrollo de la aplicación está restringido a ciertos laboratorios, por lo que cualquier problema con el acceso a dichos equipos o con el software instalado en ellos será un riesgo para nuestro proyecto.

A pesar de que en alguna ocasión los técnicos de los laboratorios han sido reacios a facilitarnos el acceso a algún laboratorio, en general no hemos tenido excesivas dificultades ni ha supuesto un problema estructural que amenazase el desarrollo del proyecto.

7. Problemas de accesibilidad al director de proyecto

El director de proyecto tiene unos horarios limitados de atención a los alumnos. Se puede producir la circunstancia de que, debido a nuestros horarios, nos sea difícil establecer reuniones con él, o bien que debido a algún tipo de percance, el director de proyecto no esté disponible durante una larga temporada.

Salvo dos semanas en las que el director del proyecto se vio obligado a desplazarse a Japón (aunque el desarrollo ya estaba completado), el director del proyecto nos ha dedicado todo el tiempo que ha sido necesario y más.

8. Problemas para combinar horarios

Los integrantes del grupo tenemos horarios y asignaturas distintas en ambos cuatrimestres. Conseguir solapar horarios de tal manera que podamos reunirnos todos los miembros del grupo el tiempo suficiente es un problema que tendremos que afrontar.

A pesar de tener horarios dispares, hemos conseguido encontrar siempre bastante tiempo para el trabajo en grupo.

9. Problemas por exceso de carga lectiva

Los integrantes del grupo deberán compaginar el desarrollo del proyecto con su respectiva carga lectiva. Un exceso de la misma supondrá una menor dedicación al proyecto o la imposibilidad de cumplir los plazos establecidas.

No consideramos haber sufrido ningún exceso de carga lectiva, lo que no significa que no hayamos empleado muchas horas en la facultad trabajando. Sin embargo, debido a una excelente planificación y a un buen reparto de tareas, nunca nos hemos sentido angustiados.

10. Caída del repositorio donde se almacenan las versiones

No se puede evitar la caída del servidor de versiones ya que está contratado a una empresa externa. Esto supondrá la imposibilidad de acceder a dichas versiones para continuar con el trabajo.

El servidor de Google Code siempre ha estado disponible.

CAPÍTULO 4
RESULTADOS Y CONCLUSIONES

4.1. RESULTADOS

Como resultado del desarrollo de este proyecto se ha obtenido la aplicación CPMP, software especializado en el campo de la astronomía.

CPMP está compuesto por un sistema de gestión de catálogos de estrellas muy versátil, capacitado incluso para mostrar placas fotográficas de las mismas por pantalla y trabajar con cualquier formato de catálogo, así como por un interfaz de desarrollo de scripts que constituye una herramienta potente y flexible para realizar todo tipo de operaciones de minería de datos sobre los catálogos.

Un ejemplo de este tipo de operaciones de minería de datos es la que ha permitido el descubrimiento de 83 estrellas dobles nuevas:

RA DEC	Mags	PA	SEP	DATE
00 32 19.099 -21 50 33.69	11.7 12.29	208.45	49.78	2000.783
00 55 33.213 -43 16 11.73	12.24 12.63	134.53	28.20	1999.713
01 00 52.512 -18 56 57.08	11.44 11.57	220.88	25.25	1998.626
01 52 27.571 -49 31 25.56	9.04 9.16	207.56	42.41	1999.812
02 04 18.761 -70 59 40.88	10.48 11.77	103.88	24.05	1999.894
02 05 27.334 +38 20 57.02	12.29 13.07	130.16	31.88	1998.810
03 10 41.550 -20 06 41.54	7.62 10.61	313.69	59.79	1998.878
03 24 54.681 -43 12 55.77	9.86 11.62	192.09	26.63	1999.648
03 46 09.569 -41 12 22.33	9.25 11.48	257.08	66.28	1999.629
04 23 48.192 -76 43 09.45	11.13 11.29	265.20	45.21	1998.840
05 01 36.173 -44 49 49.05	7.55 10.58	265.53	50.77	1999.722
05 23 39.978 -38 18 48.09	11.82 12.20	193.43	24.06	1999.173
05 57 24.758 -40 23 51.78	8.15 11.28	350.10	45.10	1999.190

RA DEC	Mags	PA	SEP	DATE
06 15 01.664 -63 20 38.82	11.71 12.57	219.59	15.59	1998.947
06 38 17.669 +18 28 24.58	11.44 11.79	63.63	55.81	1997.898
07 03 08.446 -73 50 13.91	11.12 11.34	140.15	51.35	2000.167
07 08 55.244 -11 23 29.58	11.22 12.32	31.21	48.15	1999.138
09 33 19.911 -07 11 24.75	6.25 10.78	19.14	57.81	1999.048
09 51 08.004 -18 39 31.43	7.37 10.83	115.95	50.68	1998.322
10 14 38.104 -13 33 29.10	9.04 10.34	177.99	28.62	1999.299
10 15 08.028 -65 26 11.04	11.15 11.47	304.57	31.11	2000.230
10 16 15.352 -17 11 13.12	10.67 11.78	282.61	41.57	1998.234
10 32 03.297 -30 28 05.49	11.46 11.78	29.56	14.49	1999.223
10 47 27.741 -11 54 08.72	9.83 10.46	48.64	27.76	1998.256
11 29 03.259 -38 17 05.77	9.48 11.35	109.61	38.72	1999.272
11 35 52.047 -40 40 36.43	11.85 12.18	247.31	20.08	1999.275
11 53 22.088 -67 07 05.56	10.78 11.88	119.24	36.11	2001.121
12 05 25.156 +17 17 21.69	7.63 10.04	311.55	43.20	1998.033
12 13 30.463 -48 47 46.64	8.80 10.57	347.83	49.99	1999.357
12 34 19.535 -35 22 46.48	10.46 11.63	162.81	44.30	1999.258
12 35 15.748 -09 10 57.84	10.91 10.95	142.98	28.20	1999.089
12 35 43.067 -03 00 58.10	8.49 9.79	280.18	60.70	1999.149
12 36 16.407 -79 31 34.45	10.95 11.34	254.57	14.99	2000.102
13 17 35.429 -11 57 01.34	11.04 12.70	344.71	24.29	1999.138
13 53 54.390 -07 45 44.87	11.30 11.84	216.02	18.76	1999.163
14 08 01.294 -13 16 09.22	11.84 12.31	180.53	13.24	1999.299
14 12 31.776 -30 06 17.68	11.48 11.77	216.48	31.32	1999.262
14 35 32.025 -35 26 39.17	11.44 11.72	211.37	41.19	2000.310

RA DEC	Mags	PA	SEP	DATE
14 37 23.205 -66 50 27.83	9.95 10.47	305.18	27.02	2000.258
14 53 52.152 -63 53 53.17	9.28 11.02	83.26	41.56	2000.223
14 55 28.254 -56 48 55.15	9.85 11.00	265.59	26.29	2000.146
14 58 31.045 -27 24 06.18	10.79 11.48	91.00	15.09	1998.486
15 04 08.091 -26 23 26.83	10.97 11.56	322.81	26.24	1998.486
15 07 12.834 -41 41 31.12	9.36 9.46	286.51	8.64	1999.374
15 13 59.433 -58 37 15.68	9.88 10.30	319.51	45.32	1999.431
16 31 42.851 +70 55 59.84	8.22 11.45	312.31	39.95	1999.398
16 37 35.305 +69 19 17.25	9.07 10.76	124.95	99.40	1999.399
17 01 49.674 +14 42 27.70	10.37 10.63	12.39	19.29	1999.158
17 40 21.442 +05 43 37.44	11.19 11.98	342.16	34.69	2000.404
17 54 51.203 +28 51 38.93	11.27 12.36	244.01	41.01	2000.204
17 56 59.673 -46 06 31.92	10.91 10.94	207.38	11.21	1999.551
17 59 53.975 -45 17 20.72	10.94 11.67	192.86	14.26	1999.551
18 07 28.944 +00 29 27.19	11.23 11.54	355.05	49.82	1999.548
18 13 05.162 +18 40 45.60	8.37 10.50	260.54	34.30	2000.209
18 21 26.185 -15 22 18.08	9.72 11.08	26.29	18.82	1999.333
18 22 44.038 -40 44 59.30	10.32 10.73	159.99	12.69	2000.427
18 27 24.762 +21 51 53.42	10.22 11.99	159.94	26.67	2000.242
18 38 36.531 +49 00 42.13	9.42 10.72	260.52	47.47	1998.478
18 41 25.436 -44 32 30.63	10.78 11.14	53.29	36.30	2000.474
18 45 04.604 -23 15 07.22	8.45 11.24	93.96	25.15	1998.478
18 54 43.138 -50 07 46.85	9.18 11.72	280.23	27.51	1999.726
19 01 33.281 -24 08 28.08	9.20 11.43	83.49	31.00	1999.262
19 07 06.084 -14 04 09.97	8.64 10.38	317.61	20.69	2000.247

RA DEC	Mags	PA	SEP	DATE
20 06 03.948 -41 37 36.45	11.49 11.68	8.99	23.71	1999.505
20 33 53.250 -27 10 17.31	9.35 11.97	39.94	52.00	2000.561
20 36 05.730 -67 05 22.53	10.50 11.32	107.53	24.53	2000.542
20 46 51.514 -49 28 39.62	10.71 11.39	197.77	27.90	1999.710
21 10 18.359 -13 04 05.84	11.13 11.15	246.88	16.80	1999.483
21 16 13.422 -40 40 51.94	11.30 12.07	151.91	31.23	1999.691
21 29 36.229 -44 13 50.10	10.36 10.71	261.71	90.54	1999.633
21 36 58.092 -35 53 03.02	7.35 8.60	265.34	78.45	2000.562
21 54 22.594 -44 09 46.37	11.10 11.90	73.80	17.96	1999.718
22 08 27.535 -57 06 52.51	11.08 11.45	327.83	26.33	2000.543
22 09 42.632 -33 45 15.41	9.28 10.25	278.32	49.90	1999.560
22 32 09.402 -13 35 51.81	7.72 9.71	93.97	41.94	1998.481
22 33 45.700 +61 45 26.85	9.94 10.90	222.22	38.29	1999.746
22 41 49.606 +59 47 35.64	9.03 11.02	104.02	47.92	1999.741
22 47 55.497 +03 36 07.26	11.28 11.35	154.24	23.20	2000.608
22 53 55.679 -37 09 40.50	10.22 10.59	313.30	54.44	1999.734
22 54 19.523 +30 22 18.31	10.46 11.40	233.61	14.46	1998.473
23 28 08.467 -02 26 53.36	8.05 8.43	226.37	57.33	1998.730
23 37 40.086 +00 46 36.37	10.63 12.3	156.89	34.18	2000.658

Tanto la aplicación como el descubrimiento de estas estrellas han dado lugar a dos publicaciones en el campo de la astronomía incluidas en el Apéndice 3 de este documento:

Título: *New Common Proper-Motion Pairs from the PPMX Catalogue*

Autor: Rafael Caballero, Blanca Collado, Antonio Fernández, Sara Pozuelo

Revista: Journal of Double Star Observations, Vol. 6 No. 3 July 1, 2010

Accesible: *http://www.jdso.org/*

Descripción: Se presentan 83 nuevas estrellas dobles

Título: *Una aplicación para descubrir nuevos pares de estrellas con movimiento propio común mediante técnicas de minería de datos*

Autor: Blanca Collado, Antonio Fernández, Sara Pozuelo

Revista: El Observador de Estrellas Dobles. ISSN 1989-3582

Accesible: Pendiente de publicación.

Descripción: Presentación del programa CPMP

4.2. TRABAJO FUTURO

Una vez finalizado el desarrollo de la aplicación, podemos enunciar varias líneas de trabajo que podrían mejorar el alcance funcional de la misma:

- Aumentar la interacción del sistema con la plataforma Aladin. Nuestra aplicación proporciona al usuario la posibilidad de mostrar a través de Aladin placas fotográficas de la estrella que se desee con un simple click de ratón. Sin embargo, Aladin puede proporcionar muchas más funcionalidades, como calcular la distancia entre las dos estrellas o mostrar datos de catálogos que incluyan a la estrella elegida.
- Actualmente, CPMP trabaja únicamente con el motor de bases de datos relacional MySQL. Sería interesante expandir la capa de integración a otros motores como Access u Oracle. Durante el desarrollo de la aplicación se ha tenido muy en cuenta esta posibilidad, construyendo un marco para que dicha extensión sea lo más sencilla posible.

Además de utilizar CPMP para descubrir nuevas estrellas dobles, la potencia de los scripts capacita a los usuarios para trabajar en las siguientes líneas de investigación:

- Encontrar entradas posiblemente duplicadas en los catálogos actuales.
- Elaborar listas de estrellas con características determinadas para programas observacionales.

4.3. CONCLUSIONES

Desde un primer momento, el equipo de desarrollo se planteó un par de cuestiones que han acabado siendo cruciales tanto en la elección del proyecto como en su elaboración. ¿Qué podemos aportar nosotros en el ámbito de la informática? y ¿qué puede aportar el desarrollo de un proyecto de esta magnitud a nuestra formación? Desde luego no eran dudas excepcionales pero, sin embargo, su respuesta fue definiendo el camino que hemos recorrido para llegar hasta este punto.

Decíamos en la introducción de este documento que la informática se había convertido con el devenir de los años en una herramienta indispensable en prácticamente todos los campos de la ciencia. Sectores como el de la bioquímica y la medicina fusionados con la informática o, más concretamente, con el tratamiento automatizado de la información, habían emergido con fuerza permitiendo no solo a investigadores más especializados, sino también a simples profesionales, el uso de herramientas que o bien facilitaban su trabajo o bien constituían un potente instrumento con el que alcanzar cotas que hasta el momento resultaban imposibles.

Cuando el director del proyecto nos expuso las características de esta aplicación, no nos cupo la menor duda de que nos hallábamos ante un caso similar. Era evidente que en un campo de la ciencia como es la astronomía se hacía ya uso de modernos equipos y software muy especializado, sin embargo, estábamos hablando de recursos costosos que raramente llegaban a aquellos profesionales o aficionados que no estuviesen en la vanguardia de la ciencia. Por otro lado, la labor de este último conjunto de personas, sino pionera, si era necesaria para poder analizar y exprimir la ingente cantidad de datos que se generaban diariamente desde los distintos observatorios astronómicos y centros de investigación.

CPMP nació con el objetivo de paliar en la medida de lo posible esta situación, proporcionando a la comunidad astronómica una herramienta simple e intuitiva, pero al mismo tiempo lo suficientemente potente y flexible como para ser verdaderamente útil. En este sentido, creemos haber superado los objetivos marcados. CPMP no es ya únicamente un programa para encontrar estrellas dobles, sino que se ha acabado convirtiendo también en un completo y versátil

gestor de datos, preparado para trabajar con cualquier formato de catálogos, mostrar placas fotográficas de las estrellas y que, además, cuenta con una interfaz de desarrollo de scripts que aumenta el alcance de la aplicación en función a los conocimientos y necesidades del usuario.

Por otro lado, también consideramos importante evaluar la aportación de este proyecto a nuestro perfil académico. Sin lugar a dudas, el desarrollo de CPMP nos ha permitido poner en práctica muchos de los conocimientos troncales que hemos adquirido durante la carrera, conocimientos que hemos conseguido transformar en soluciones prácticas a problemas reales. Para ser más concretos y, haciendo un poco de recapitulación a partir de todo lo expuesto a lo largo de este documento, ofrecemos los siguientes ejemplos:

- El "parseador" que nos permite importar catálogos a partir de ficheros de texto fue diseñado utilizando una aproximación al concepto de autómata finito determinista, que aprendimos en la asignatura "Teoría de Autómatas y Lenguajes Formales"(2º curso).
- El intérprete usado para el módulo de desarrollo de scripts se implementó aplicando gramáticas y conocimientos impartidos en "Procesadores de lenguaje" (4º curso).
- Para aumentar la velocidad de funcionalidades como cruzar catálogos, se utilizaron algoritmos de divide y vencerás estudiados en "Metodología y Tecnología de la Programación" (3er curso).
- Tanto la idea de utilizar hilos para evitar el bloqueo de la aplicación en aquellos casos de uso que demandan mayor carga del sistema, como la solución a los problemas de concurrencia que estos nos generaron, fueron fruto de "Sistemas Operativos" (3er curso).
- El debate sobre como representar los tipos de datos que maneja CPMP, así como la decisión de usar tablas hash para almacenar los diferentes hilos de ejecución, estuvo fundamentado en los conocimientos adquiridos en "Estructura de Datos y de la Información" (2º curso).
- La implementación en Java, así como la correcta distribución de roles y responsabilidades entre los distintos componentes del sistema, hubiese

sido imposible sin haber cursado “Programación Orientada a Objetos” (2º curso).

- “Ingeniería del Software” (4º curso) nos preparó para estructurar un plan de proyecto, obtener una especificación de requisitos, aplicar el concepto de arquitectura multicapa y usar patrones de diseño, conocimientos sin los cuales el desarrollo de la aplicación hubiese sido caótico.
- En último lugar, es obvio que sin haber cursado “Bases de Datos y Sistemas de Información” (optativa) el desarrollo de esta aplicación no hubiese sido posible.

Asimismo, no solo hemos puesto en práctica conocimientos adquiridos a lo largo de la carrera, sino que también hemos profundizado notablemente en el conocimiento de la plataforma Java, nos hemos aficionado a la astronomía y hemos disfrutado de una gran experiencia de trabajo en equipo, la cual nos gustaría resumir citando al escritor irlandés George Bernard Shaw:

> “[…] Si tú tienes una idea y yo tengo una idea y las intercambiamos, entonces ambos tendremos dos ideas."

APÉNDICES

APÉNDICE 1: MANUAL

¿Qué es CPMP?

CPMP es un software especializado en el descubrimiento de nuevos pares de estrellas con movimiento propio común, estrellas dobles, utilizando técnicas de almacenamiento heterogéneo y minería de datos.

Dichas técnicas son aplicadas sobre catálogos de estrellas de extensas dimensiones previamente descargados del Servicio Astronómico VizieR, por lo que CPMP ofrece al usuario una herramienta simple, intuitiva y automatizada, para gestionar dichos catálogos de forma unificada. Debido a la necesidad de manejar los mismos catálogos en el laborioso proceso de minería de datos, la aplicación precisa de un gestor de bases de datos para la persistencia de los mismos.

¿Qué se necesita para ejecutar CPMP?

El primer requisito que necesita CPMP es la instalación de la máquina virtual de Java de Sun Microsystems (JRE v. 1.6.0 o posterior). En caso de no tenerla instalada, es posible descargarla desde la página web de java: http://www.java.com/es/download/manual.jsp .

El segundo y último requisito es tener instalado el sistema de bases de datos MySql versión 5.1.40, que puede ser descargado desde http://www.mysql.com/downloads/mysql/ de forma gratuita.

CPMP ha sido verificada y trabaja de forma correcta en entornos Windows XP, Windows Vista, Windows 7 y distribuciones GNU/Linux (Ubuntu y Fedora).

Instalación y ejecución

Al ser una aplicación multiplataforma, ejecutable en distintos sistemas operativos, no se ha realizado un instalador de la misma para ninguna arquitectura específica.

La aplicación se proporciona de forma auto-ejecutable en un fichero JAR (Java ARchives). Por tanto la instalación simplemente consiste en descargar el archivo CPMP.jar y la ejecución en hacer doble click sobre él.

Cómo usar CPMP

La primera ventana que se observa al ejecutar CPMP es la correspondiente a la conexión con el esquema de bases de datos. Como ya se ha explicado, la aplicación necesita de un gestor de Bases de Datos para la persistencia de los catálogos de estrellas.

.

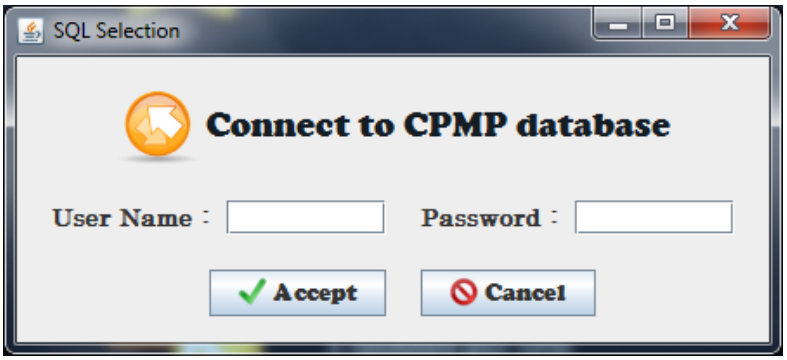


Figura 1. Ventana de conexión con la base de datos

La ventana de conexión solicita introducir un nombre de usuario y contraseña de MySQL. Si es la primera vez que se ejecuta CPMP, se creará de forma automática un esquema de Bases de Datos denominado CPMP en el disco del usuario. De esta forma todos los datos serán almacenados en el mencionado esquema y, en las sucesivas ejecuciones de la aplicación, se podrá seguir trabajando con dichos datos ya que, automáticamente, en cada inicio de conexión CPMP se conecta a su esquema determinado.

Al iniciar conexión correctamente se mostrara el siguiente mensaje:

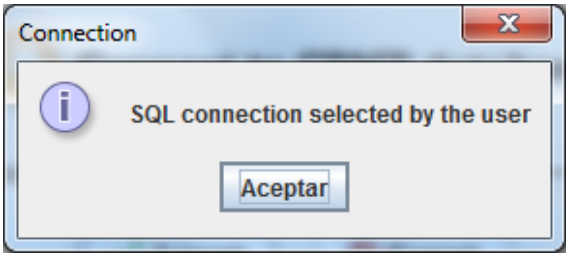


Figura 2. Mensaje de conexión correcta

Y se podrá visualizar la ventana de inicio de la aplicación, con las opciones más importantes de CPMP (*Figura 3*).

Figura 3. Ventana de inicio

En caso de que el nombre o la contraseña no se introduzcan de forma correcta el usuario será informado con otro mensaje:

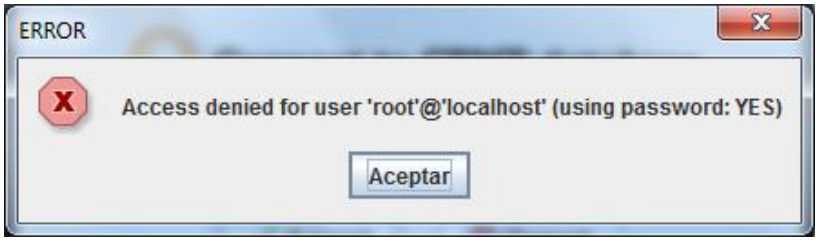


Figura 4. Mensaje de conexión errónea

Si se cancela la conexión a la base de datos CPMP, el marco de inicio será distinto al mostrado cuando se realiza la conexión de forma correcta. En este caso, solo serán accesibles dos opciones: la opción de conexión a la base de datos, para permitir reintentar el *login*, y la opción de visualizar la ayuda. (*Figura 5*).

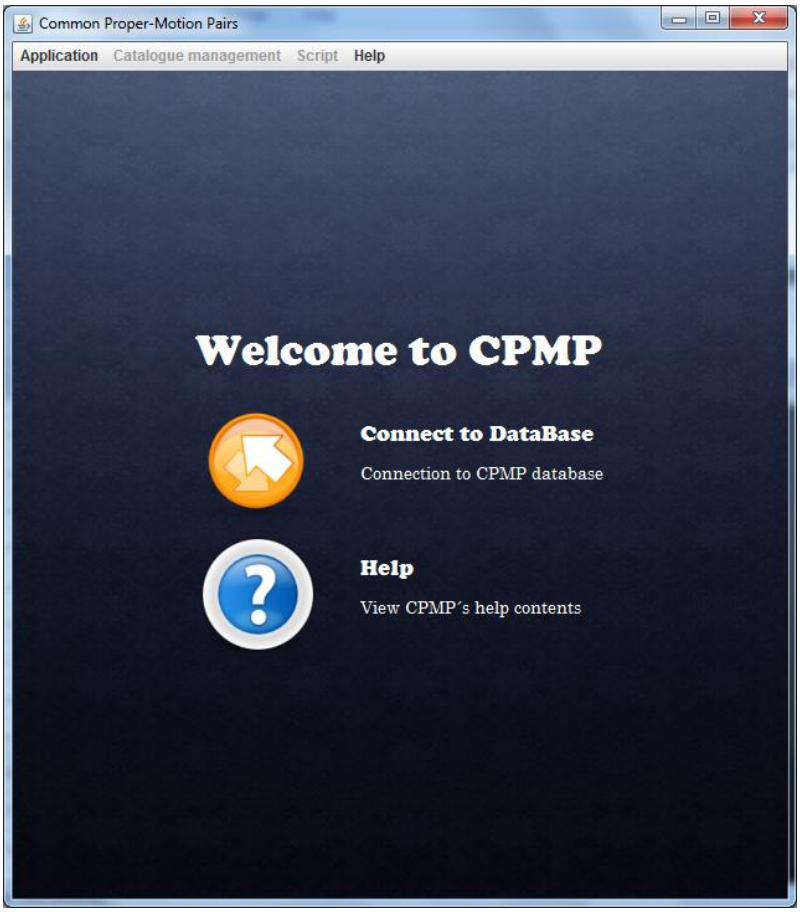


Figura 5. Ventana de inicio al cancelar la conexión con la base de datos

Las funcionalidades de CPMP se dividen muy claramente en dos grupos, Gestión de Catálogos y Script, como se puede observar en la barra de herramientas de la aplicación.

Gestión de Catálogos

La primera opción de Gestión de Catálogos es, **Import New Catalogue**  , es la operación básica antes de poder comenzar la labor de minería de datos, ya que es la que permitirá cargar los datos de las estrellas individuales en la aplicación. Dichos datos deberán ser catálogos previamente descargados de VizieR (*servicio de catálogos astronómicos proporcionado por el Centre de Données astronomiques de Strasbourg (CDS)*) en formato de texto y con los campos separados por ";". (*Ver* *Apéndice 2: Descargando Catálogos de VizieR*)

Al pulsar dicha opción se mostrará un cuadro de selección de fichero que permite elegir el fichero del catálogo a importar. Una vez seleccionado, se deberá introducir el nombre con el que se le identificará dentro de CPMP (*Figura 6*).

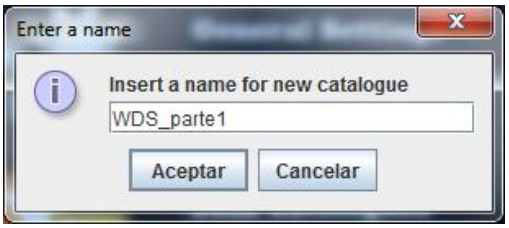


Figura 6. Insertar nombre de catálogo

Una vez pulsado el botón *Aceptar*, comenzará el proceso de importar el catálogo. Dicho proceso puede ser tedioso, puesto que el tiempo necesario para realizarlo es lineal con respecto al tamaño del fichero de entrada (que puede llegar a ocupar varios gigas de memoria). Por ello, CPMP permite añadir varios catálogos a la vez (*Figura 7*) realizando esta labor en un segundo plano y permitiendo así seguir trabajando con catálogos previamente añadidos.

Otra de las utilidades que ofrece CPMP en el proceso de importar catálogos es la posibilidad de finalizar el proceso en el momento que se desee, habiéndose guardado en la aplicación el catálogo con las filas importadas hasta ese momento.

Esta opción está disponible en el menú **Catalogue Managuement** -> **Import Catalogue** -> **Finish Import** , mostrando una nueva ventana con la lista de catálogos en proceso de importar. En dicha ventana se podrá seleccionar el catálogo que se desee finalizar pulsando el botón **Finish**.

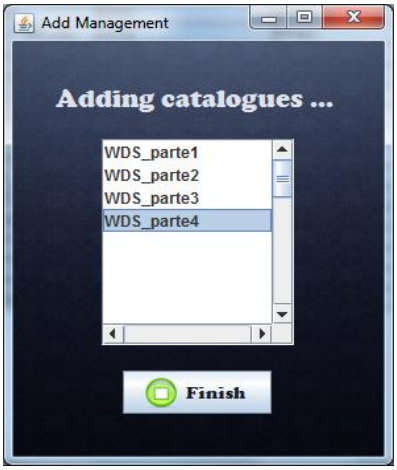


Figura 7. Lista de catálogos importándose

Según vaya finalizando el proceso de importar cada catálogo, será mostrado un informe con los siguientes datos: número de estrellas añadidas, número de estrellas no añadidas por estar duplicadas en el fichero de entrada y el tiempo requerido en realizar el proceso. *(Figura 8)*

Figura 8. Informe de resultado de un catálogo importado

Si los catálogos no han sido descargados de VizieR es decir, no tienen la cabecera correcta para ser importados, pero sí contienen los campos separados

por ";", se podrá crear una cabecera válida para ellos con la opción **Create Header for non-VizieR Catalogue** del submenú **Import Catalogue** de **Catalogue Managuement**.

Un ejemplo del contenido de un catálogo sin cabecera es el que se muestra en el siguiente cuadro de texto.

```
000140.9-432147 ;0.42062;-43.363289;37.89;39.87;12.102;6;  ;S
000854.0-463717 ;2.225293;-46.621424;35.59;33.99;11.635;7; ;S
001706.5-521427 ;4.277433;-52.24086;70.48;37.43;11.621;8;  ;S
001927.6-464437 ;4.865066;-46.743762;51.1;42.52;12.19;4;   ;S
002401.0-744942 ;6.004367;-74.82855;44.45;37.23;11.935;7;  ;S
002836.9-571038 ;7.153752;-57.177443;55.01;38.27;11.721;6; ;S
003007.3-483054 ;7.530556;-48.515222;49.91;34.45;12.345;5; ;S
003025.7-584920 ;7.607208;-58.822403;59.37;37.71;12.087;5; ;S
003507.5-564002 ;8.781487;-56.667488;38.42;36.2;11.615;6;  ;S
```

Al pulsar la opción de Crear Cabecera se muestra una nueva ventana *(Figura 9)* con una serie de pasos a seguir. El primero de ellos es seleccionar el archivo de origen donde está almacenado el catálogo. El segundo consiste en introducir el nombre y el título del catálogo. Como paso tercero se debe introducir el número de campos que tiene el catálogo (por defecto este parámetro vale dos, ya que los campos RAJ2000 y DEJ2000 son obligatorios).

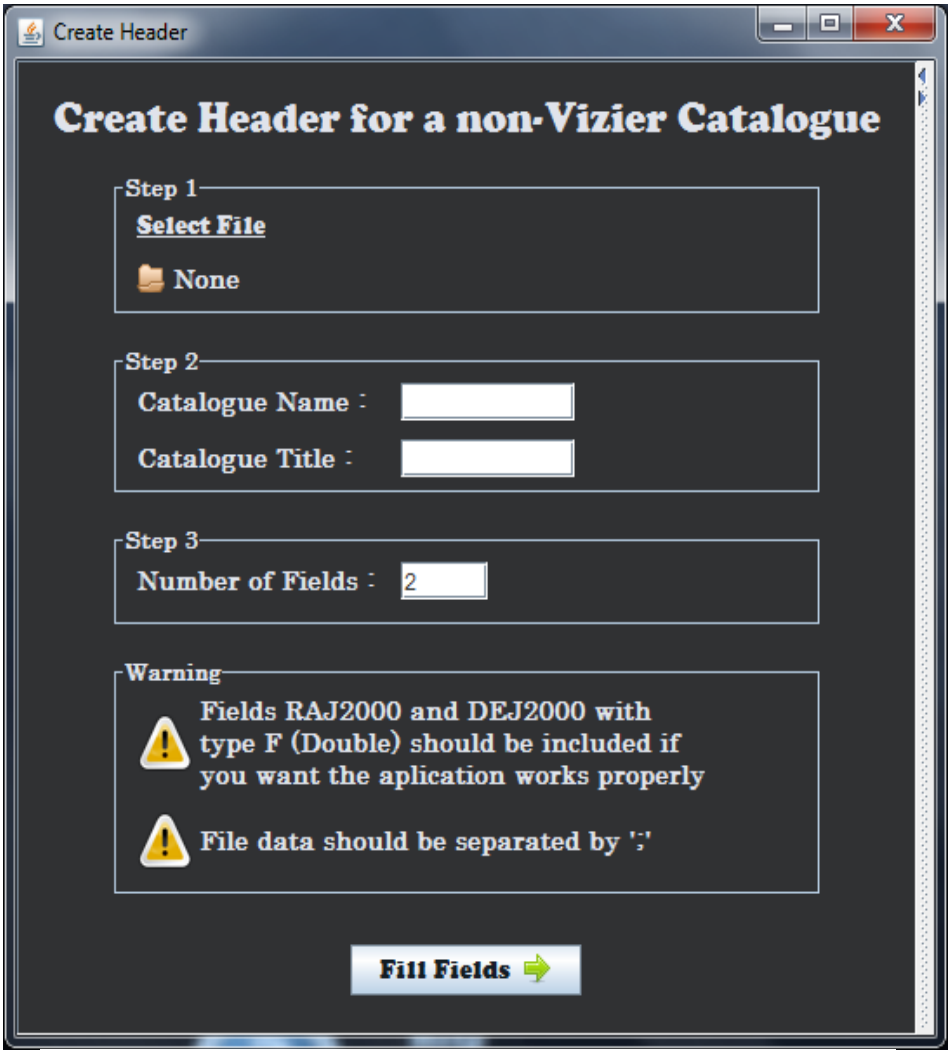


Figura 9. Ventana inicial para crear cabecera

Los requisitos antes mencionados pueden ser observados en el panel, en el apartado **Warning**. Una vez completados estos tres primeros pasos, es posible pulsar el botón **Fill Fields** sin que sea mostrado un mensaje de error. Esta acción ampliará la ventana actual con una nueva sección en la que se solicitará al usuario que rellene un formulario con el nombre y tipo de los campos.

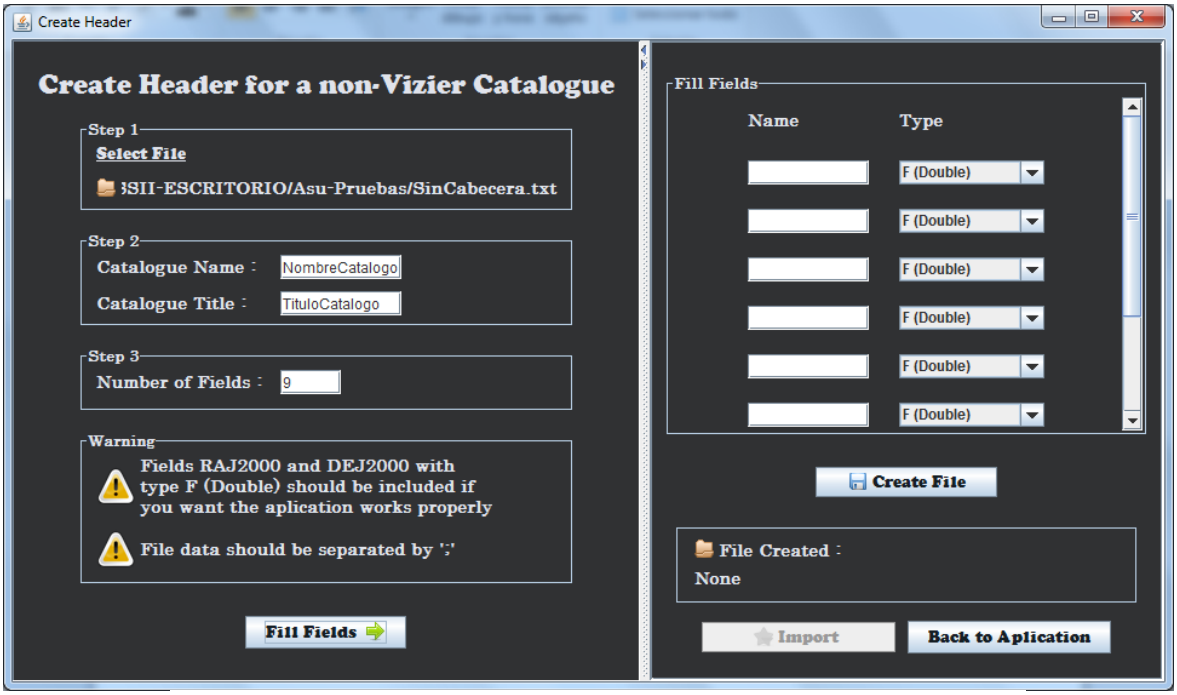


Figura 10. Ventana extendida para crear cabecera

El formulario que se muestra dentro de la sección **Fill Fields** tendrá tantos campos como se haya indicado en el paso tres del primer panel. Es necesario rellenar todos los nombres de los campos, así como sus tipos, sin olvidar que los campos RAJ2000 y DEJ2000 deben estar incluidos con tipo Double.

Una vez completados todos los campos, se deberá pulsar el botón **Create Fields** que creará un nuevo archivo con los mismos datos que el archivo de origen pero con una cabecera VizieR válida. Una vez creado el archivo, la ruta del mismo se actualizará en la sección **File Created**.

Llegado a este punto el usuario puede optar entre importar el nuevo archivo creado o volver a la aplicación.

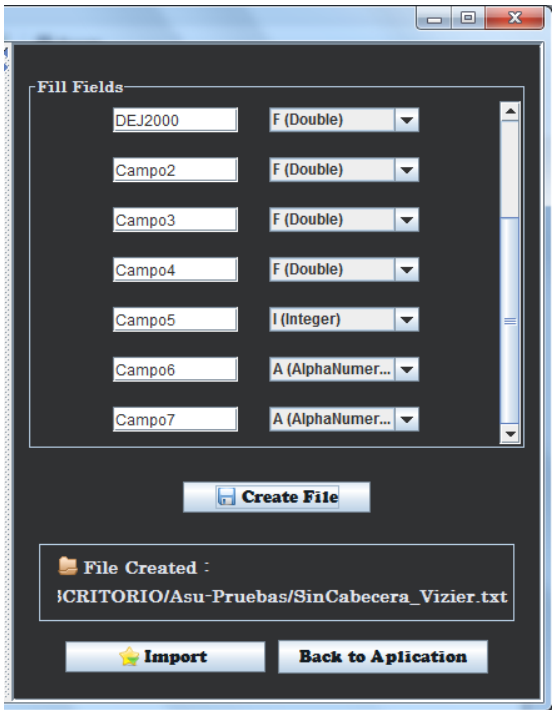


Figura 11. Crear cabecera completado

A continuación se muestra el archivo resultante de aplicar la operación crear cabecera al fichero anterior, que sólo contenía datos.

```
#Name: pruebaTitle
#Title: pruebaName
#Column uno      (A)
#Column RAJ2000  (F)
#Column DEJ2000  (F)
#Column dos      (F)
#Column tres     (F)
#Column cuatro   (F)
#Column cinco    (I)
#Column seis     (A)
#Column siete    (A)
---------------------
000140.9-432147 ;0.42062;-43.363289;37.89;39.87;12.102;6;  ;S
000854.0-463717 ;2.225293;-46.621424;35.59;33.99;11.635;7; ;S
001706.5-521427 ;4.277433;-52.24086;70.48;37.43;11.621;8;  ;S
001927.6-464437 ;4.865066;-46.743762;51.1;42.52;12.19;4;   ;S
002401.0-744942 ;6.004367;-74.82855;44.45;37.23;11.935;7;  ;S
002836.9-571038 ;7.153752;-57.177443;55.01;38.27;11.721;6; ;S
003007.3-483054 ;7.530556;-48.515222;49.91;34.45;12.345;5; ;S
```

El resto de opciones de la Gestión de Catálogos son las siguientes:

- ◊ **General Settings** (Opciones Generales).
- ◊ **Join Catalogues** (Cruzar Catálogos).
- ◊ **Divide Catalogues** (Dividir Catálogos).

Al seleccionar cualquiera de ellas cambiara la apariencia de la aplicación, convirtiéndose en una ventada dividida en dos paneles, el primero con un menú de iconos a la izquierda de la pantalla y el segundo (a la derecha) es especifico de cada opción *(Figura 12)*.

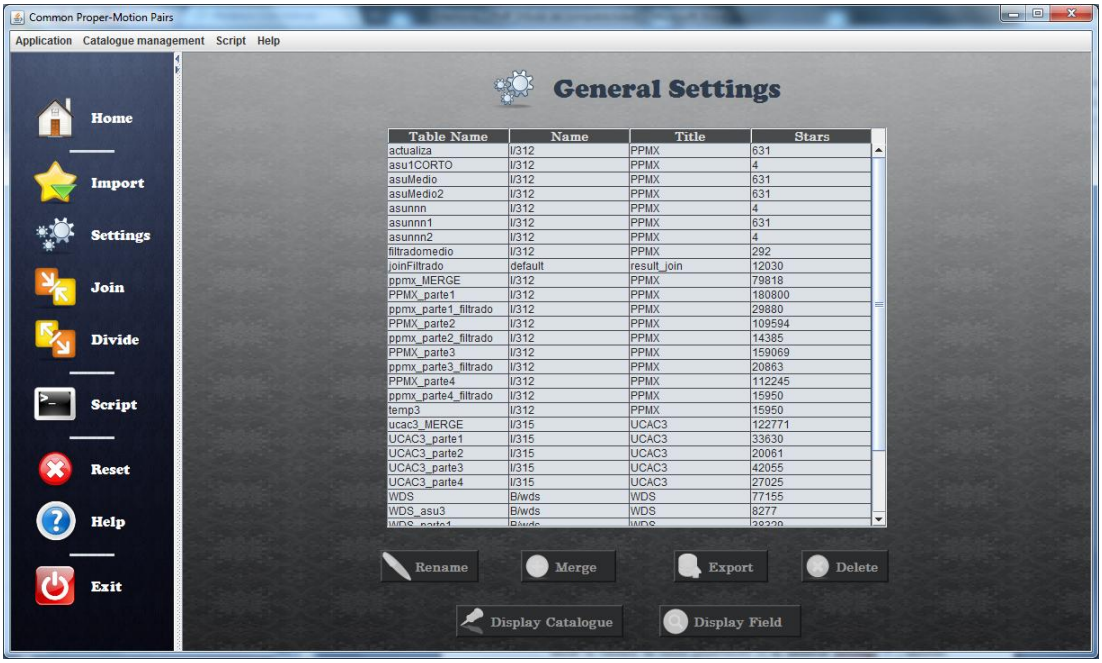


Figura 12. Panel Principal con la opción General Setting

Es posible volver al panel de inicio clikeando el icono **Home** del menú lateral.

GENERAL SETTINGS

En General Settings se permite realizar ciertas operaciones generales sobre los catálogos incluidos en la aplicación. Para ello se muestra la lista de Catálogos disponibles con la información básica que los caracteriza:

- **Table Name** : Nombre de Catálogo definido por el usuario.
- **Name** : Identificación del Catálogo por VizieR.
- **Title** : Título del Catálogo por VizieR.
- **Stars**: Número de estrellas contenidas en el Catálogo.

Se puede observar como, al seleccionar cualquiera de los catálogos de la lista, clickeando sobre ellos, se habilitan algunos de los botones ubicados debajo de la misma. Esto significa que las operaciones habilitadas se pueden aplicar sobre el catálogo seleccionado. El único botón que no se habilita al seleccionar sólo un catálogo es el correspondiente a la operación **Merge**, ya que esta operación necesita de, al menos, dos catálogos para aplicarse. Se podrán seleccionar varios catálogos pulsando la tecla *Control* del teclado mientras se marcan en la lista.

Las operaciones son las siguientes:

-  **Rename Catalogue.**
-  **Merge Catalogues.**
-  **Delete Catalogue.**
-  **Export Catalogue.**
-  **Display Catalogue.**
-  **Display Catalogue Fields.**

Rename Catalogue

Crea un nuevo catálogo clonando uno ya existente. El usuario introducirá mediante teclado el nombre con el que persistirá este catálogo dentro de la aplicación.

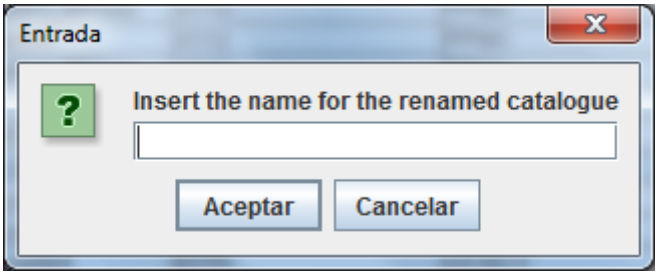


Figura 13. Ventana de petición de Nombre del Catálogo

Se indicará un mensaje de error en caso de que el nombre elegido ya exista en la aplicación. Se puede observar cómo se añade de forma inmediata el nuevo catálogo a la lista de catálogos mostrada cuando la operación se realiza de forma satisfactoria.

Delete Catalogue

Elimina el catálogo o los catálogos seleccionados de la base de datos. El usuario debe confirmar su decisión antes de que la operación sea efectiva y se actualiza la lista de catálogos al finalizar ésta.

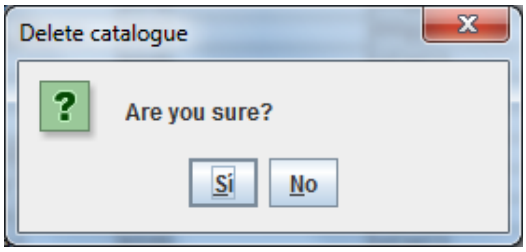


Figura 14. Mensaje de confirmación al eliminar un catálogo

Merge Catalogues

Une dos o más catálogos, tantos como hayan sido seleccionados, en un nuevo catálogo. Para ello se muestra una ventana de petición del nombre del nuevo catálogo.

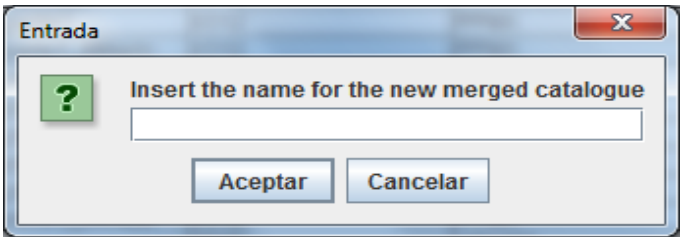


Figura 15. Ventana de petición del nombre del nuevo catálogo de unión

Los catálogos deben tener el mismo formato. Si se decide unir catálogos que no cumplen dicha restricción, se mostrará un mensaje de error.

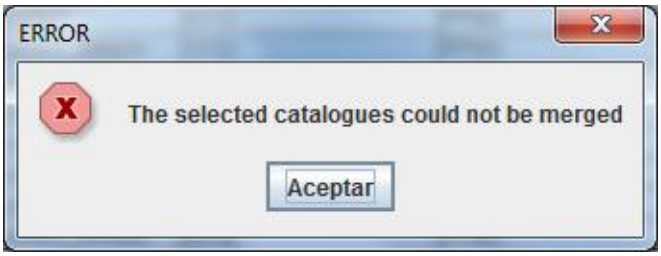


Figura 16. Mensaje de error al unir catálogos

Si la operación se realiza de forma correcta, se muestra un informe con los datos del resultado de la unión.

Figura 17. Informe del resultado de la unión de varios catálogos

Export Catalogue

Exporta el catálogo seleccionado en un fichero de texto. Este catálogo empezará por una cabecera VizieR válida generada por la aplicación.

Para ello se solicita seleccionar el directorio donde será almacenado dicho fichero y el nombre con el que se desea guardar.

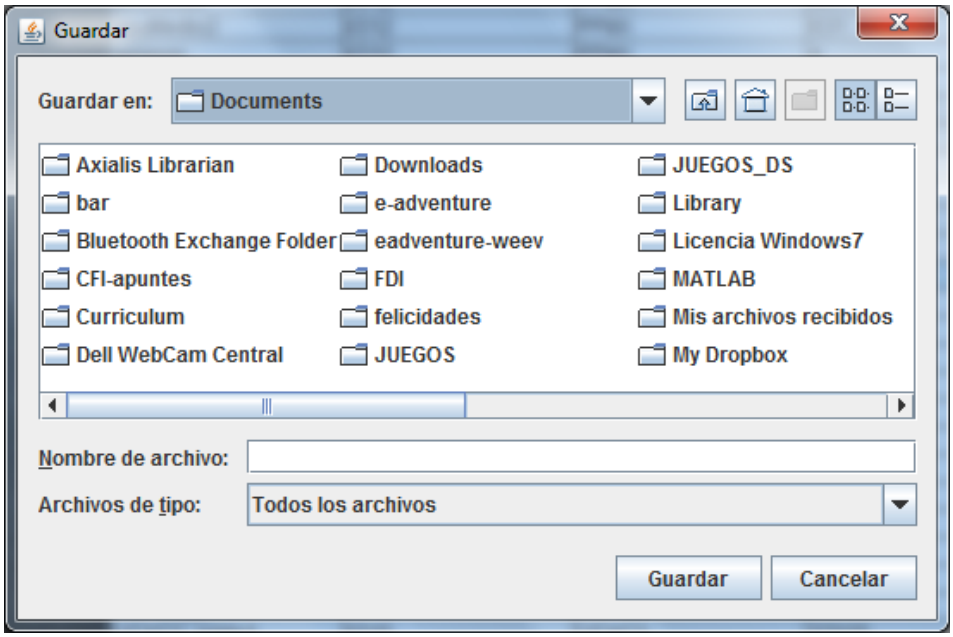


Figura 18. Selección de directorio

Display Catalogue Fields

Muestra la lista de campos del catálogo seleccionado en una nueva ventana, especificando el nombre y el tipo de cada campo. Si se hace click en el botón derecho del ratón sobre cualquier campo, se muestra un menú contextual con la opción **Delete**. Si se selecciona dicha opción se eliminara el campo el elegido del catálogo, actualizándose de forma inmediata la lista de campos.

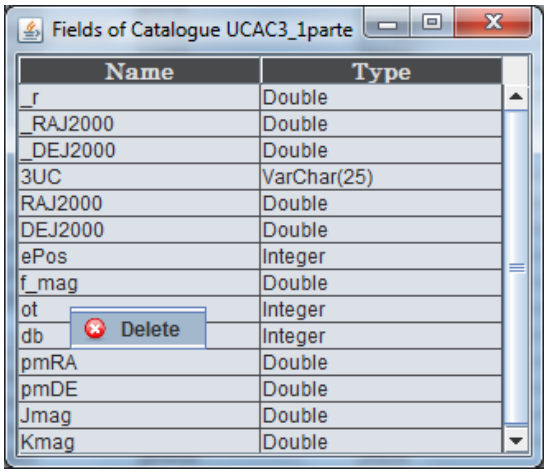
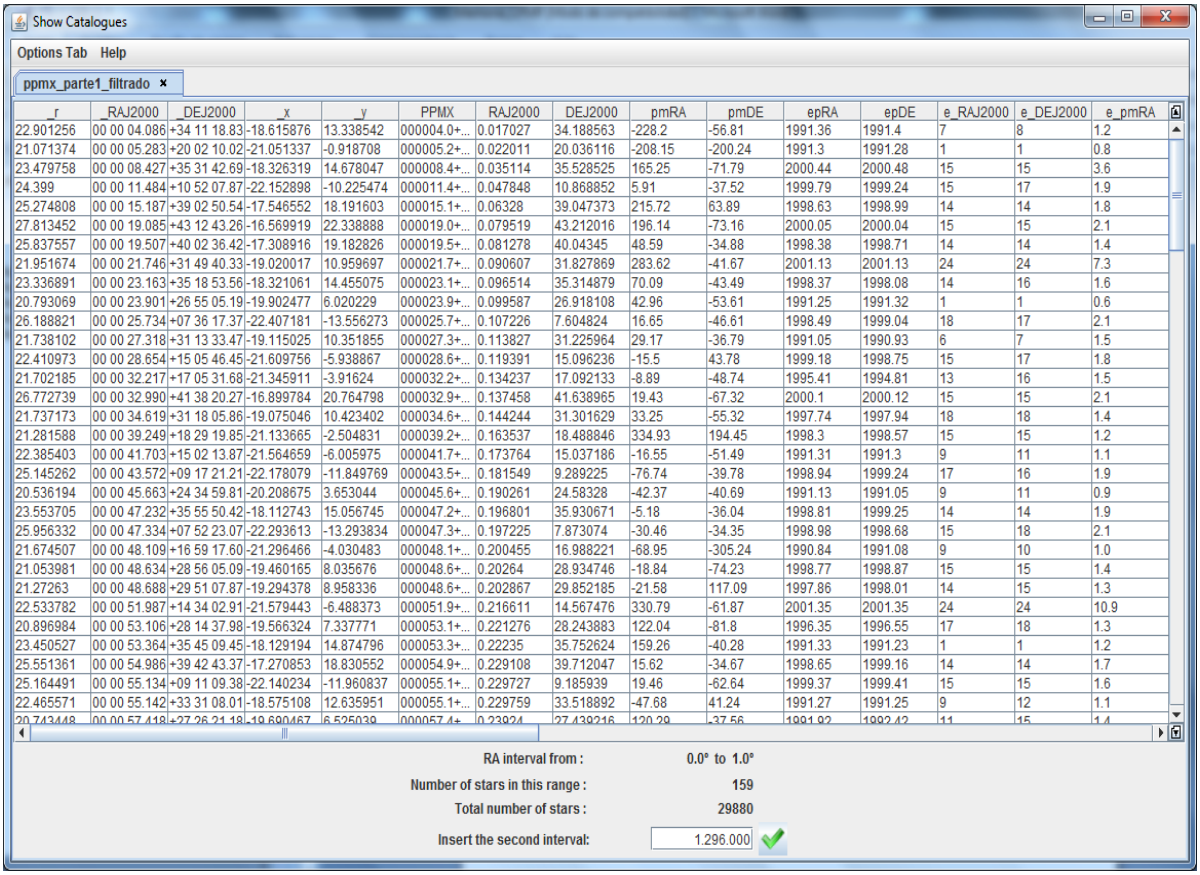


Figura 19. Lista de Campos de un catálogo seleccionado y menu contextual con la opción de eleminar un determinado campo

Display Catalogue

Al seleccionar esta opción se mostrará una nueva ventana con el contendido del catálogo elegido.



_r	_RAJ2000	_DEJ2000	_x	_y	PPMX	RAJ2000	DEJ2000	pmRA	pmDE	epRA	epDE	e_RAJ2000	e_DEJ2000	e_pmRA
22.901256	00 00 04.086	+34 11 18.83	-18.615876	13.338542	000004.0+...	0.017027	34.188563	-228.2	-56.81	1991.36	1991.4	7	8	1.2
21.071374	00 00 05.283	+20 02 10.02	-21.051337	-0.918708	000005.2+...	0.022011	20.036116	-208.15	-200.24	1991.3	1991.28	1	1	0.8
23.479758	00 00 08.427	+35 31 42.69	-18.326319	14.678047	000008.4+...	0.035114	35.528525	165.25	-71.79	2000.44	2000.48	15	15	3.6
24.399	00 00 11.484	+10 52 07.87	-22.152898	-10.225474	000011.4+...	0.047848	10.868852	5.91	-37.52	1999.79	1999.24	15	17	1.9
25.274808	00 00 15.187	+39 02 50.54	-17.546552	18.191603	000015.1+...	0.06328	39.047373	215.72	63.89	1998.63	1998.99	14	14	1.8
27.813452	00 00 19.085	+43 12 43.26	-16.569919	22.338888	000019.0+...	0.079519	43.212016	196.14	-73.16	2000.05	2000.04	15	15	2.1
25.837557	00 00 19.507	+40 02 36.42	-17.308916	19.182826	000019.5+...	0.081278	40.04345	48.59	-34.88	1998.38	1998.71	14	14	1.4
21.951674	00 00 21.746	+31 49 40.33	-19.020017	10.959697	000021.7+...	0.090607	31.827869	283.62	-41.67	2001.13	2001.13	24	24	7.3
23.336891	00 00 23.163	+35 18 53.56	-18.321061	14.455075	000023.1+...	0.096514	35.314879	70.09	-43.49	1998.37	1998.08	14	16	1.6
20.793069	00 00 23.901	+26 55 05.19	-19.902477	6.020229	000023.9+...	0.099587	26.918108	42.96	-53.61	1991.25	1991.32	1	1	0.6
26.188821	00 00 25.734	+07 36 17.37	-22.407181	-13.556273	000025.7+...	0.107226	7.604824	16.65	-46.61	1998.49	1999.04	18	17	2.1
21.738102	00 00 27.318	+31 13 33.47	-19.115025	10.351855	000027.3+...	0.113827	31.225964	29.17	-36.79	1991.05	1990.93	6	7	1.5
22.410973	00 00 28.654	+15 05 46.45	-21.609756	-5.938867	000028.6+...	0.119391	15.096236	-15.5	43.78	1999.18	1998.75	15	17	1.8
21.702185	00 00 32.217	+17 05 31.68	-21.345911	-3.91624	000032.2+...	0.134237	17.092133	-8.89	-48.74	1995.41	1994.81	13	16	1.5
26.772739	00 00 32.990	+41 38 20.27	-16.899784	20.764798	000032.9+...	0.137458	41.638965	19.43	-67.32	2000.1	2000.12	15	15	2.1
21.737173	00 00 34.619	+31 18 05.86	-19.075046	10.423402	000034.6+...	0.144244	31.301629	33.25	-55.32	1997.74	1997.94	18	18	1.4
21.281588	00 00 39.249	+18 29 19.85	-21.133665	-2.504831	000039.2+...	0.163537	18.488846	334.93	194.45	1998.3	1998.57	15	15	1.2
22.385403	00 00 41.703	+15 02 13.87	-21.564659	-6.005975	000041.7+...	0.173764	15.037186	-16.55	-51.49	1991.31	1991.3	9	11	1.1
25.145262	00 00 43.572	+09 17 21.21	-22.178079	-11.849769	000043.5+...	0.181549	9.289225	-76.74	-39.78	1998.94	1999.24	17	16	1.9
20.536194	00 00 45.663	+24 34 59.81	-20.208675	3.653044	000045.6+...	0.190261	24.58328	-42.37	-40.69	1991.13	1991.05	9	11	0.9
23.553705	00 00 47.232	+35 55 50.42	-18.112743	15.056745	000047.2+...	0.196801	35.930671	-5.18	-36.04	1998.81	1999.25	14	14	1.9
25.956332	00 00 47.334	+07 52 23.07	-22.293613	-13.293834	000047.3+...	0.197225	7.873074	-30.46	-34.35	1998.98	1998.68	15	18	2.1
21.674507	00 00 48.109	+16 59 17.60	-21.296466	-4.030483	000048.1+...	0.200455	16.988221	-68.95	-305.24	1990.84	1991.08	9	10	1.0
21.053981	00 00 48.634	+28 56 05.09	-19.460165	8.035676	000048.6+...	0.20264	28.934746	-18.84	-74.23	1998.77	1998.87	15	15	1.4
21.27263	00 00 48.688	+29 51 07.87	-19.294378	8.958336	000048.6+...	0.202867	29.852185	-21.58	117.09	1997.86	1998.01	14	15	1.3
22.533782	00 00 51.987	+14 34 02.91	-21.579443	-6.488373	000051.9+...	0.216611	14.567476	330.79	-61.87	2001.35	2001.35	24	24	10.9
20.896984	00 00 53.106	+28 14 37.98	-19.566324	7.337771	000053.1+...	0.221276	28.243883	122.04	-81.8	1996.35	1996.55	17	18	1.3
23.450527	00 00 53.364	+35 45 09.45	-18.129194	14.874796	000053.3+...	0.22235	35.752624	159.26	-40.28	1991.33	1991.23	1	1	1.2
25.551361	00 00 54.986	+39 42 43.37	-17.270853	18.830552	000054.9+...	0.229108	39.712047	15.62	-34.67	1998.65	1999.16	14	14	1.7
25.164491	00 00 55.134	+09 11 09.38	-22.140234	-11.960837	000055.1+...	0.229727	9.185939	19.46	-62.64	1999.37	1999.41	15	15	1.6
22.465571	00 00 55.142	+33 31 08.01	-18.575108	12.635951	000055.1+...	0.229759	33.518892	-47.68	41.24	1991.27	1991.25	9	12	1.1
20.743448	00 00 57.418	+27 26 21.18	-19.690467	6.525039	000057.4+...	0.23924	27.439216	120.29	-37.56	1991.92	1992.42	11	15	1.4

Figura 20. Ventana con el contenido de los catálogos

Cada catálogo seleccionado será mostrado en una pestaña diferente en la misma ventana, indicando en cada pestaña el nombre de dicho catálogo.

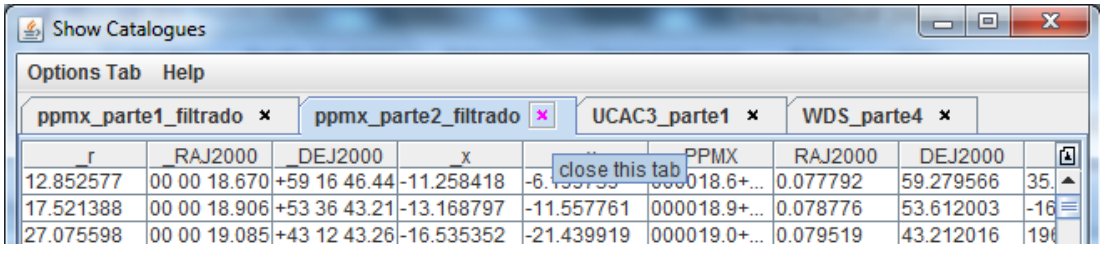


Figura 21. Pestañas de la Ventana Show Catalogues

Se puede observar que, además del nombre del catálogo en cada pestaña, se muestra un aspa que nos permite cerrar la pestaña seleccionada. Además se

ofrecen distintas opciones para gestionar el aspecto de las pestañas, que se encuentran en el menú **Options Tab**:

- ◊ **Use Tab Components**: casilla de verificación, por defecto activada, que permite mostrar el aspa de cierre de pestaña o no mostrarlo.
- ◊ **Set ScrollLayout**: casilla de verificación, por defecto desactivada, que al ser activada no permite al usuario moverse por las pestañas a través de una barra de desplazamiento.
- ◊ **Reset Tabs**: botón que permite cerrar todas las pestañas a la vez.

La otra opción del menú es **Help,** que muestra la ayuda específica de esta ventana.

Los catálogos, debido a su gran extensión, se muestran por partes divididas en intervalos dependientes de la Ascensión Recta y Declinación (coordenadas que caracterizan a cada estrella) de las estrellas contenidas.

Al seleccionar la opción **Show catalogue** de un determinado catálogo, se muestra un primer conjunto de estrellas contenidas en dicho catálogo que están comprendidas en el intervalo de Ascensión Recta entre 0 y 1 grados.

La información correspondiente al intervalo mostrado en cada momento está siempre visible en la parte inferior del panel.

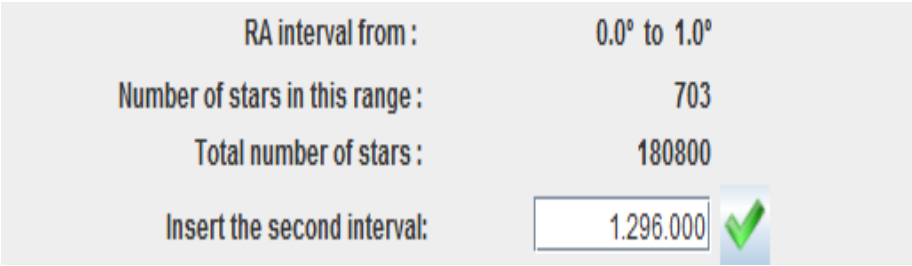
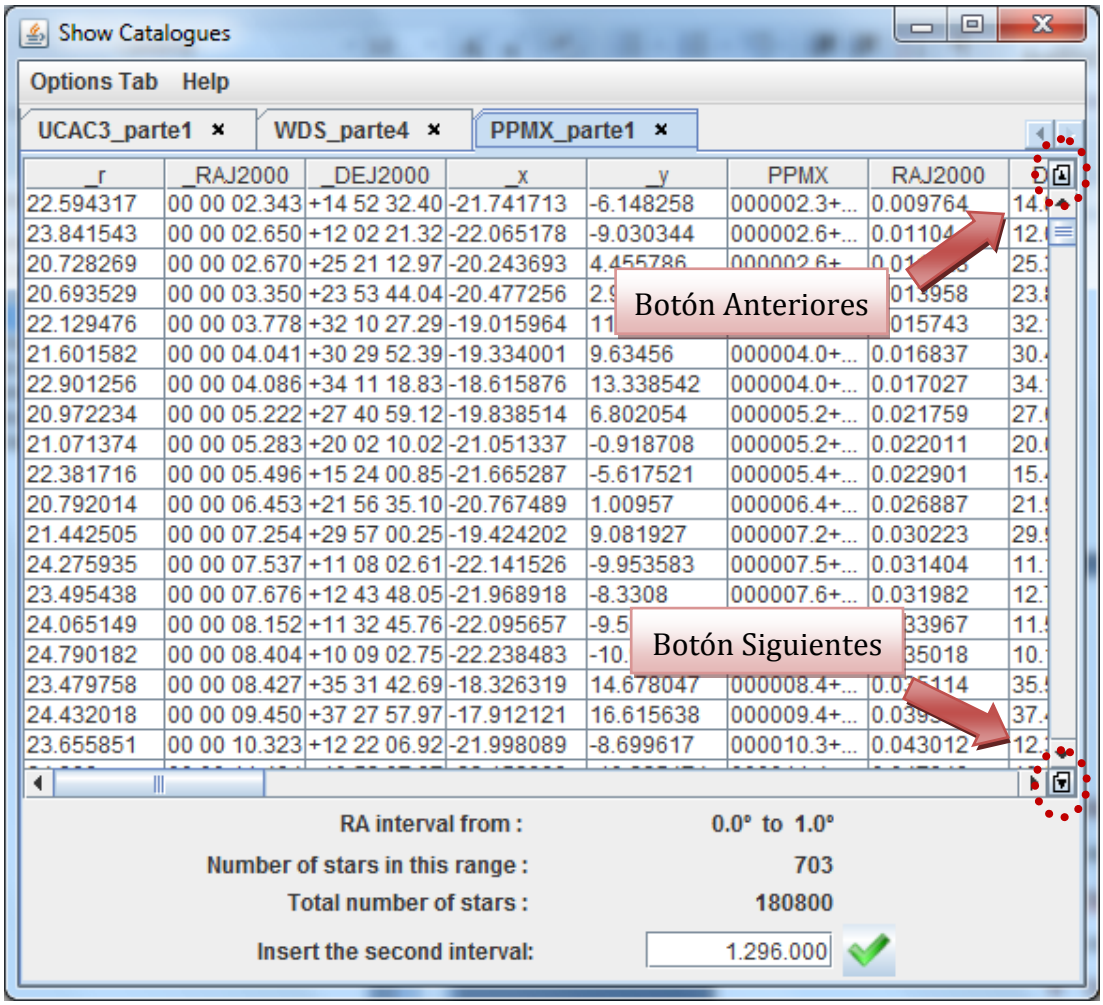


Figura 22. Información de un catálogo mostrado

- ◊ **RA interval**: corresponde al Intervalo de Ascensión Recta mostrado.
- ◊ **Number of stars in this range**: es el número de estrellas contenidas en dicho intervalo
- ◊ **Total number of stars**: es el número total de estrellas del catálogo.

Para visualizar el resto del contenido del catálogo, es posible navegar por él gracias a dos botones, anterior y siguiente. Ambos muestran el contenido anterior o siguiente en función del intervalo actual y del rango elegido para dichos intervalos (por defecto son intervalos de 1 grado).



_r	_RAJ2000	_DEJ2000	_x	_y	PPMX	RAJ2000
22.594317	00 00 02.343	+14 52 32.40	-21.741713	-6.148258	000002.3+...	0.009764
23.841543	00 00 02.650	+12 02 21.32	-22.065178	-9.030344	000002.6+...	0.01104
20.728269	00 00 02.670	+25 21 12.97	-20.243693	4.455786	000002.6+...	0.01
20.693529	00 00 03.350	+23 53 44.04	-20.477256	2.9		013958
22.129476	00 00 03.778	+32 10 27.29	-19.015964	11		015743
21.601582	00 00 04.041	+30 29 52.39	-19.334001	9.63456	000004.0+...	0.016837
22.901256	00 00 04.086	+34 11 18.83	-18.615876	13.338542	000004.0+...	0.017027
20.972234	00 00 05.222	+27 40 59.12	-19.838514	6.802054	000005.2+...	0.021759
21.071374	00 00 05.283	+20 02 10.02	-21.051337	-0.918708	000005.2+...	0.022011
22.381716	00 00 05.496	+15 24 00.85	-21.665287	-5.617521	000005.4+...	0.022901
20.792014	00 00 06.453	+21 56 35.10	-20.767489	1.00957	000006.4+...	0.026887
21.442505	00 00 07.254	+29 57 00.25	-19.424202	9.081927	000007.2+...	0.030223
24.275935	00 00 07.537	+11 08 02.61	-22.141526	-9.953583	000007.5+...	0.031404
23.495438	00 00 07.676	+12 43 48.05	-21.968918	-8.3308	000007.6+...	0.031982
24.065149	00 00 08.152	+11 32 45.76	-22.095657	-9.5		33967
24.790182	00 00 08.404	+10 09 02.75	-22.238483	-10.		35018
23.479758	00 00 08.427	+35 31 42.69	-18.326319	14.678047	000008.4+...	0.0
24.432018	00 00 09.450	+37 27 57.97	-17.912121	16.615638	000009.4+...	0.039
23.655851	00 00 10.323	+12 22 06.92	-21.998089	-8.699617	000010.3+...	0.043012

Figura 23. Como navegar por el contenido del catálogo : Botones Anterior y Siguiente

Por ejemplo, en la *Figura 23*, se puede observar las estrellas contenidas en el intervalo de ascensión recta entre 0 y 1 grado del catálogo PPMX_parte1 y, como se ha especificado anteriormente, por defecto se muestran intervalos de rango 1 grado. Al pulsar el botón siguiente, el contenido mostrado será el correspondiente a la ascensión recta de la última estrella del intervalo anterior ([0°, 1°]), y a sumar

1 (rango de intervalos por defecto) a dicha ascensión recta. Como se puede observar en la siguiente ilustración este intervalo es de 0.99976 a 1.99976 y el número de estrellas contenidas en este nuevo intervalo son 761.

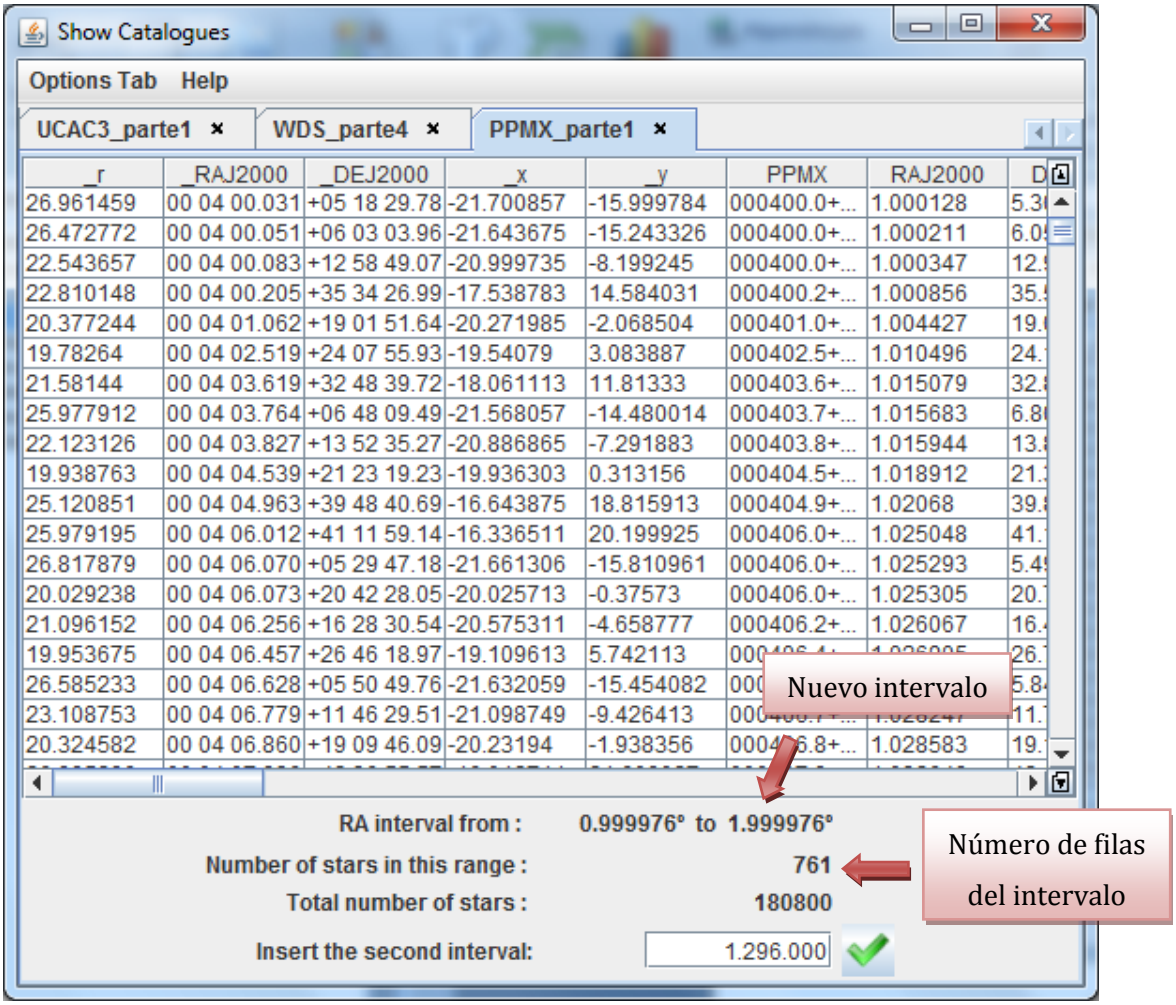


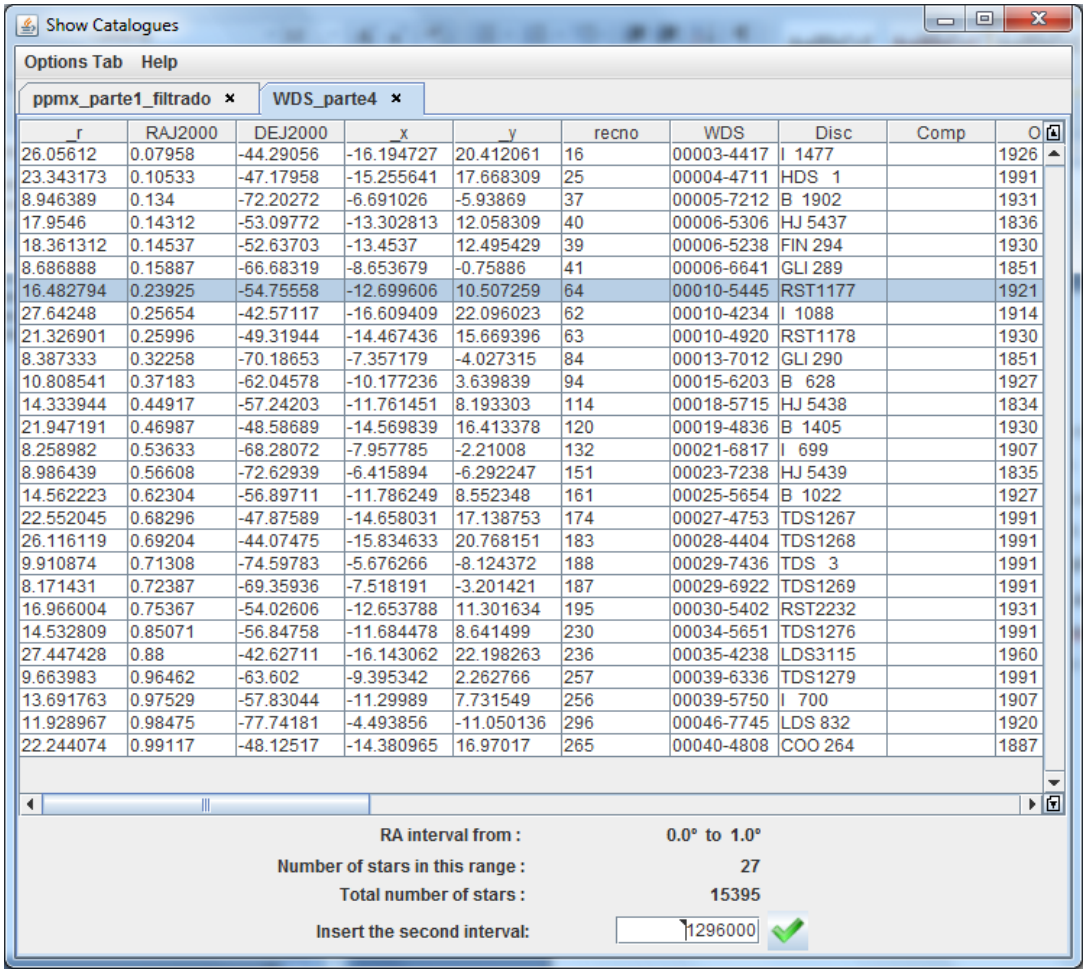
Figura 24. Ejemplo de intervalo después de pulsar el botón siguiente

De esta forma se puede navegar por todo el contenido del catálogo, que siempre se encontrará entre 0 y 360 grados.

Si el número de estrellas del catálogo no es elevado se puede incrementar el rango de intervalo mostrado, es decir, en vez de mostrar intervalos de diferencia 1° , mostrarlos de 45° por ejemplo visualizar de 45° a 90°.

Para ello se debe introducir el rango deseado en segundos en el campo de inserción correspondiente a **Insert the second interval** y después hacer click en

el botón **Ok**. Por defecto dicho campo contiene 1296000 segundos, que equivalen a 360 grados.



_r	RAJ2000	DEJ2000	_x	_y	recno	WDS	Disc	Comp	O
26.05612	0.07958	-44.29056	-16.194727	20.412061	16	00003-4417	I 1477		1926
23.343173	0.10533	-47.17958	-15.255641	17.668309	25	00004-4711	HDS 1		1991
8.946389	0.134	-72.20272	-6.691026	-5.93869	37	00005-7212	B 1902		1931
17.9546	0.14312	-53.09772	-13.302813	12.058309	40	00006-5306	HJ 5437		1836
18.361312	0.14537	-52.63703	-13.4537	12.495429	39	00006-5238	FIN 294		1930
8.686888	0.15887	-66.68319	-8.653679	-0.75886	41	00006-6641	GLI 289		1851
16.482794	0.23925	-54.75558	-12.699606	10.507259	64	00010-5445	RST1177		1921
27.64248	0.25654	-42.57117	-16.609409	22.096023	62	00010-4234	I 1088		1914
21.326901	0.25996	-49.31944	-14.467436	15.669396	63	00010-4920	RST1178		1930
8.387333	0.32258	-70.18653	-7.357179	-4.027315	84	00013-7012	GLI 290		1851
10.808541	0.37183	-62.04578	-10.177236	3.639839	94	00015-6203	B 628		1927
14.333944	0.44917	-57.24203	-11.761451	8.193303	114	00018-5715	HJ 5438		1834
21.947191	0.46987	-48.58689	-14.569839	16.413378	120	00019-4836	B 1405		1930
8.258982	0.53633	-68.28072	-7.957785	-2.21008	132	00021-6817	I 699		1907
8.986439	0.56608	-72.62939	-6.415894	-6.292247	151	00023-7238	HJ 5439		1835
14.562223	0.62304	-56.89711	-11.786249	8.552348	161	00025-5654	B 1022		1927
22.552045	0.68296	-47.87589	-14.658031	17.138753	174	00027-4753	TDS1267		1991
26.116119	0.69204	-44.07475	-15.834633	20.768151	183	00028-4404	TDS1268		1991
9.910874	0.71308	-74.59783	-5.676266	-8.124372	188	00029-7436	TDS 3		1991
8.171431	0.72387	-69.35936	-7.518191	-3.201421	187	00029-6922	TDS1269		1991
16.966004	0.75367	-54.02606	-12.653788	11.301634	195	00030-5402	RST2232		1931
14.532809	0.85071	-56.84758	-11.684478	8.641499	230	00034-5651	TDS1276		1991
27.447428	0.88	-42.62711	-16.143062	22.198263	236	00035-4238	LDS3115		1960
9.663983	0.96462	-63.602	-9.395342	2.262766	257	00039-6336	TDS1279		1991
13.691763	0.97529	-57.83044	-11.29989	7.731549	256	00039-5750	I 700		1907
11.928967	0.98475	-77.74181	-4.493856	-11.050136	296	00046-7745	LDS 832		1920
22.244074	0.99117	-48.12517	-14.380965	16.97017	265	00040-4808	COO 264		1887

Figura 25. Ejemplo de mostrar catálogo con pocas estrellas

Por ejemplo, la primera vez que se visualiza el catálogo WDS_parte4 se observa que en el intervalo de 0 a 1 grados solo contiene 27 estrellas, y que el catálogo completo solo son 15395 estrellas por lo que se decide cambiar el rango del intervalo para mostrar más. El nuevo rango elegido será de 45 ° por lo que el valor introducido es 162000 segundos. A continuación se pulsa el botón **Ok**. Se puede observar cómo se actualiza inmediatamente el contenido de la tabla con las nuevas estrellas, y la información relativa a los intervalos.

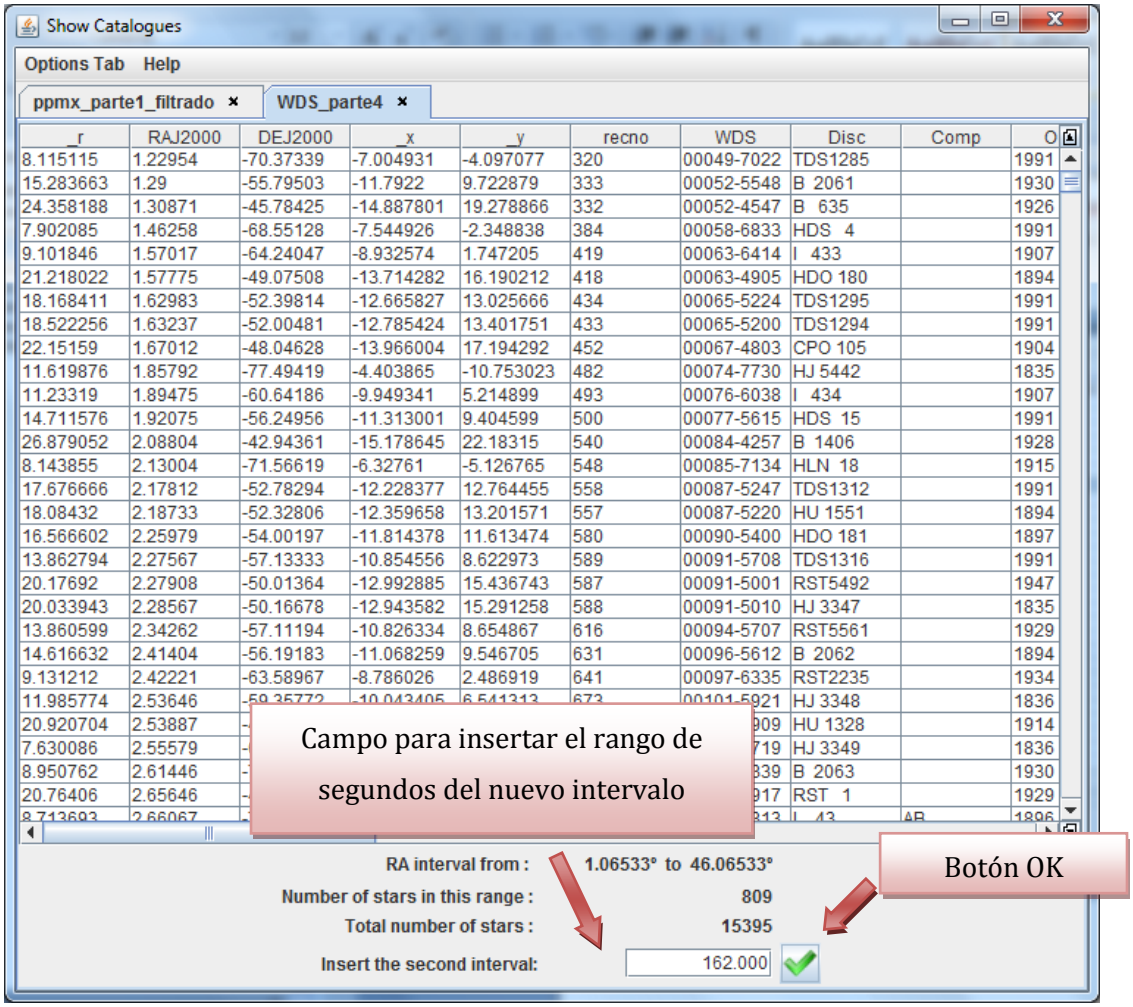
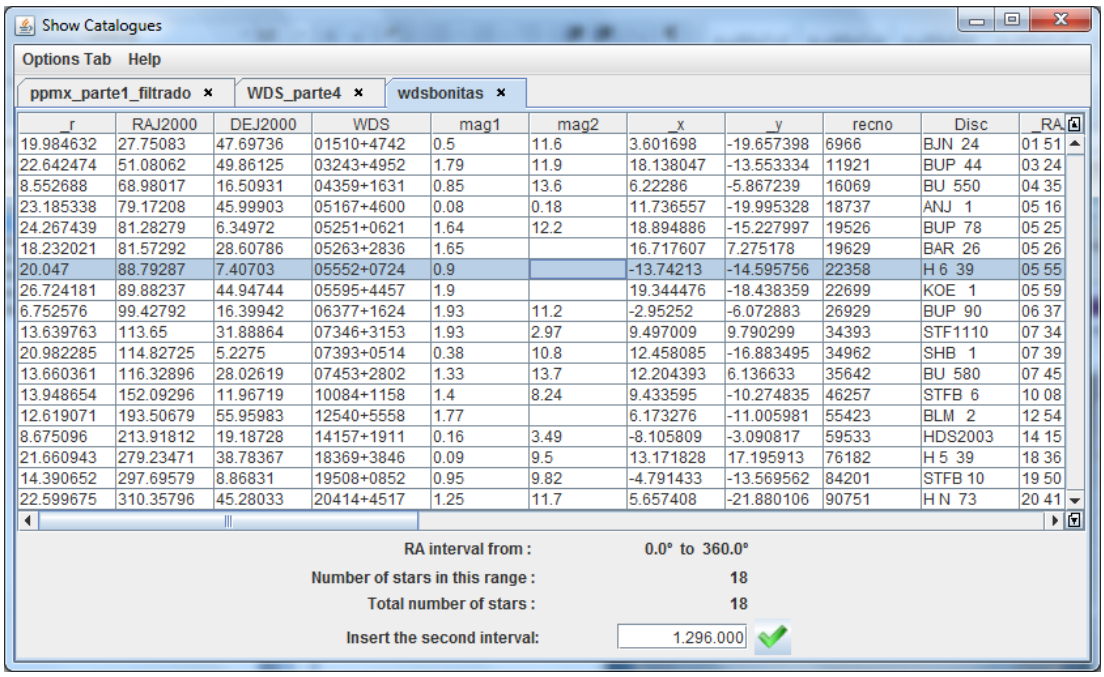


Figura 26. Incrementamos el rango de intervalo a mostrar

Además de todas las funcionalidades ya explicadas de la ventana correspondiente a **Show Catalogue**, se dispone de la opción de visualizar una estrella a través de las placas fotográficas disponibles en la aplicación online Aladin. Para ello solo se tendrá que hacer doble click en la estrella que se desea visualizar.

Por ejemplo, si se hace doble click en la estrella con AR2000 88.79287 y DEJ2000 7.40703 del catálogo WDS se mostrara en el navegador predeterminado la misma imagen que se observa en la *Figura 28*:



_r	RAJ2000	DEJ2000	WDS	mag1	mag2	_x	_y	recno	Disc	_RA
19.984632	27.75083	47.69736	01510+4742	0.5	11.6	3.601698	-19.657398	6966	BJN 24	01 51
22.642474	51.08062	49.86125	03243+4952	1.79	11.9	18.138047	-13.553334	11921	BUP 44	03 24
8.552688	68.98017	16.50931	04359+1631	0.85	13.6	6.22286	-5.867239	16069	BU 550	04 35
23.185338	79.17208	45.99903	05167+4600	0.08	0.18	11.736557	-19.995328	18737	ANJ 1	05 16
24.267439	81.28279	6.34972	05251+0621	1.64	12.2	18.894886	-15.227997	19526	BUP 78	05 25
18.232021	81.57292	28.60786	05263+2836	1.65		16.717607	7.275178	19629	BAR 26	05 26
20.047	88.79287	7.40703	05552+0724	0.9		-13.74213	-14.595756	22358	H 6 39	05 55
26.724181	89.88237	44.94744	05595+4457	1.9		19.344476	-18.438359	22699	KOE 1	05 59
6.752576	99.42792	16.39942	06377+1624	1.93	11.2	-2.95252	-6.072883	26929	BUP 90	06 37
13.639763	113.65	31.88864	07346+3153	1.93	2.97	9.497009	9.790299	34393	STF1110	07 34
20.982285	114.82725	5.2275	07393+0514	0.38	10.8	12.458085	-16.883495	34962	SHB 1	07 39
13.660361	116.32896	28.02619	07453+2802	1.33	13.7	12.204393	6.136633	35642	BU 580	07 45
13.948654	152.09296	11.96719	10084+1158	1.4	8.24	9.433595	-10.274835	46257	STFB 6	10 08
12.619071	193.50679	55.95983	12540+5558	1.77		6.173276	-11.005981	55423	BLM 2	12 54
8.675096	213.91812	19.18728	14157+1911	0.16	3.49	-8.105809	-3.090817	59533	HDS2003	14 15
21.660943	279.23471	38.78367	18369+3846	0.09	9.5	13.171828	17.195913	76182	H 5 39	18 36
14.390652	297.69579	8.86831	19508+0852	0.95	9.82	-4.791433	-13.569562	84201	STFB 10	19 50
22.599675	310.35796	45.28033	20414+4517	1.25	11.7	5.657408	-21.880106	90751	H N 73	20 41

RA interval from : 0.0° to 360.0°
Number of stars in this range : 18
Total number of stars : 18
Insert the second interval: 1.296.000

Figura 27. Pulsamos doble click sobre una fila de la tabla mostrada

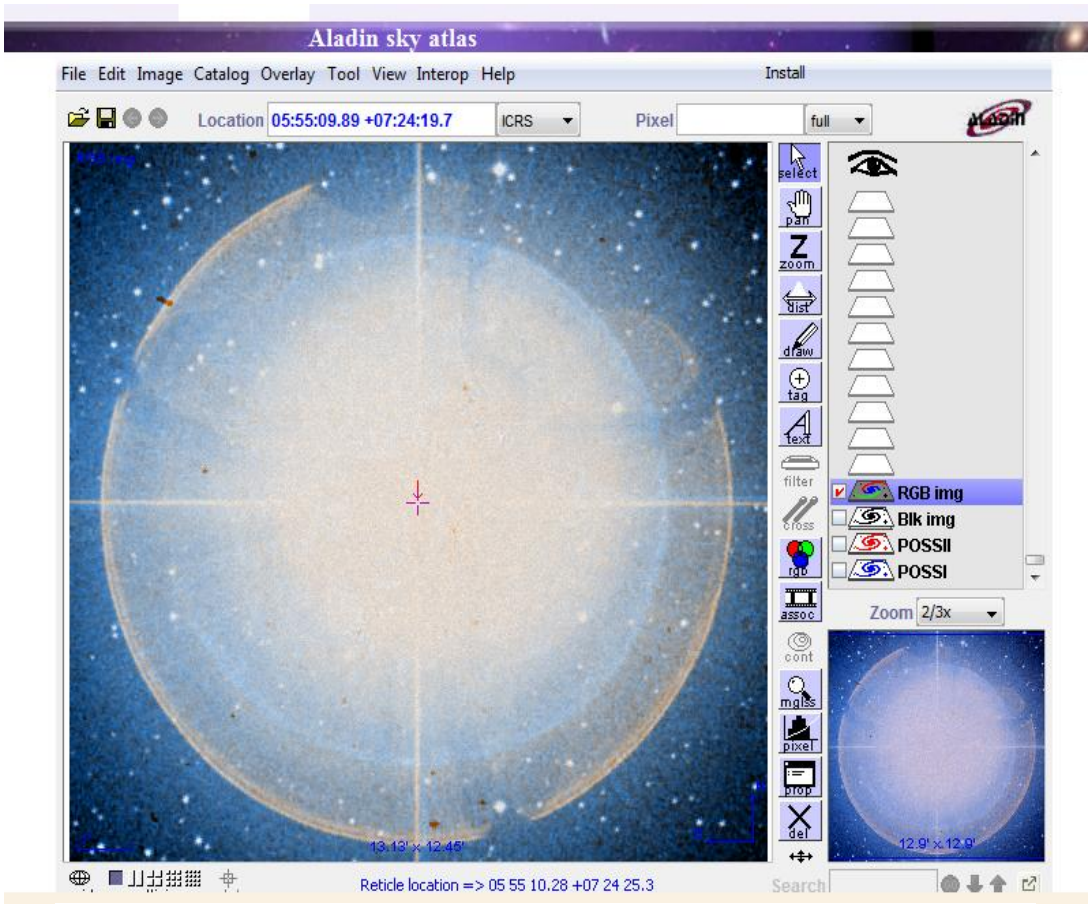


Figura 28. Placa fotográfica de Aladin de una estrella seleccionada

JOIN CATALOGUES

El panel Join Catalogues permite cruzar dos catálogos, creando una nueva tabla (catálogo de pares) que contiene por cada fila un par de estrellas cuya separación es menor que un parámetro en segundos. Esta operación permite comenzar con la propia labor de minería de datos, ya que una de las aplicaciones más comunes de esta operación es cruzar una tabla consigo misma para obtener una lista de pares candidatos, primer paso en el proceso de minería de datos.

Para ello se muestra la lista de Catálogos no cruzados por duplicado, para que se seleccione un catálogo de cada lista. Además hay que insertar el intervalo de separación para el cruce, en segundos, por defecto 3600 segundos, y el nombre del catálogo resultado de la operación. Después de rellenar los campos hay que pulsar el botón **Aceptar**.

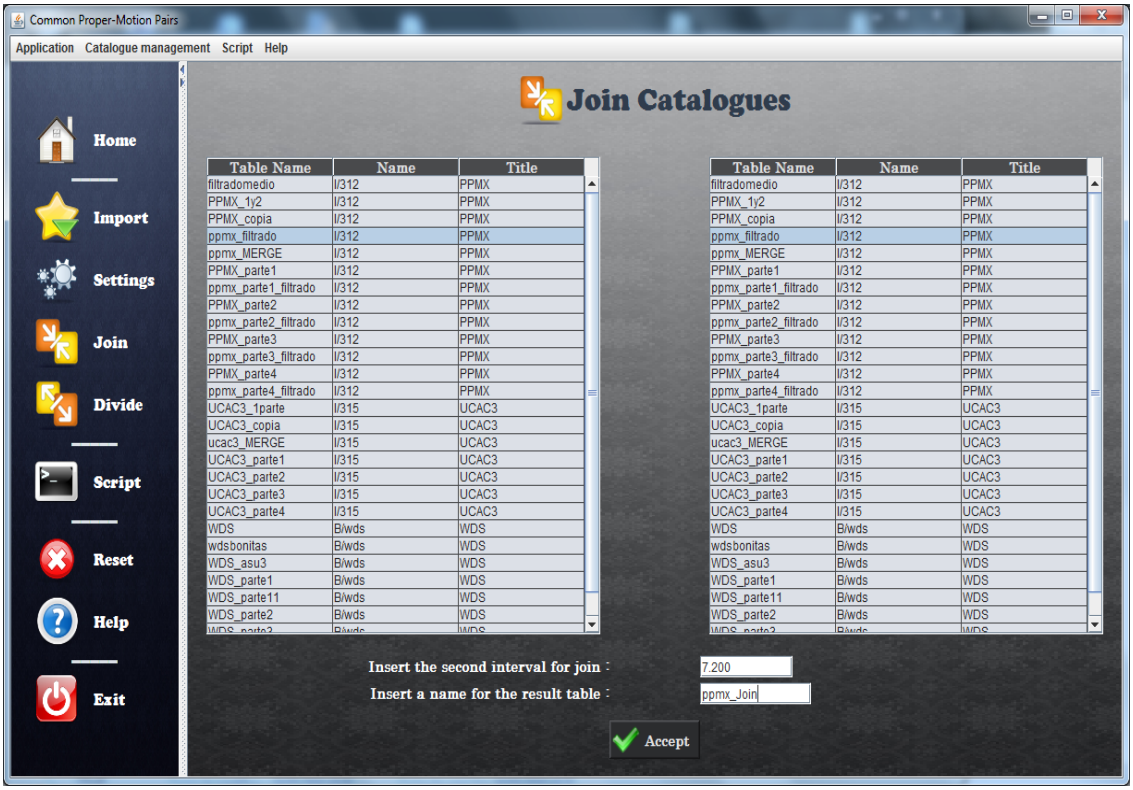


Figura 29. Panel Principal con la opción Join Catalogues

Al finalizar la operación se muestra un informe con algunos datos relativos al proceso de cruce (*Figura 30*) y el contenido del catálogo resultado en una nueva pestaña de la vista **Show Catalogue**.

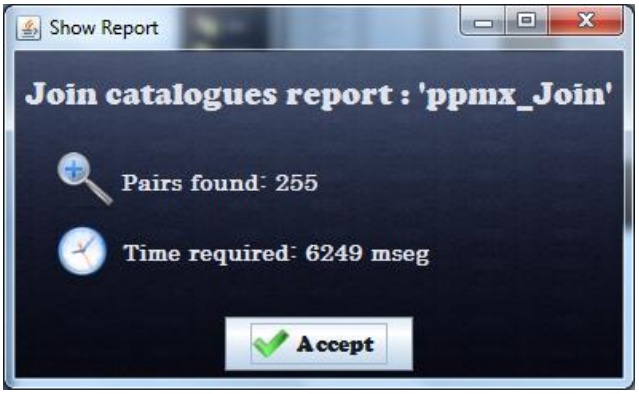


Figura 30. Informe del proceso de resultado de un Join

Si no se inserta correctamente alguno de los dos campos se muestra un mensaje de error. Por ejemplo, si se pulsa el botón **Aceptar** sin haber introducido un nombre para el nuevo catálogo se observará el siguiente mensaje:

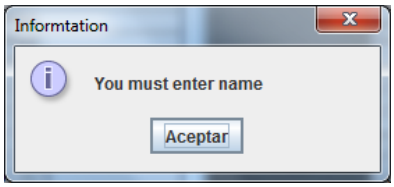


Figura 31. Mensaje de información del Panel Join

DIVIDE CATALOGUES

El panel Divide Catalogues permite realizar la división en dos catálogos de un catálogo previamente cruzado.

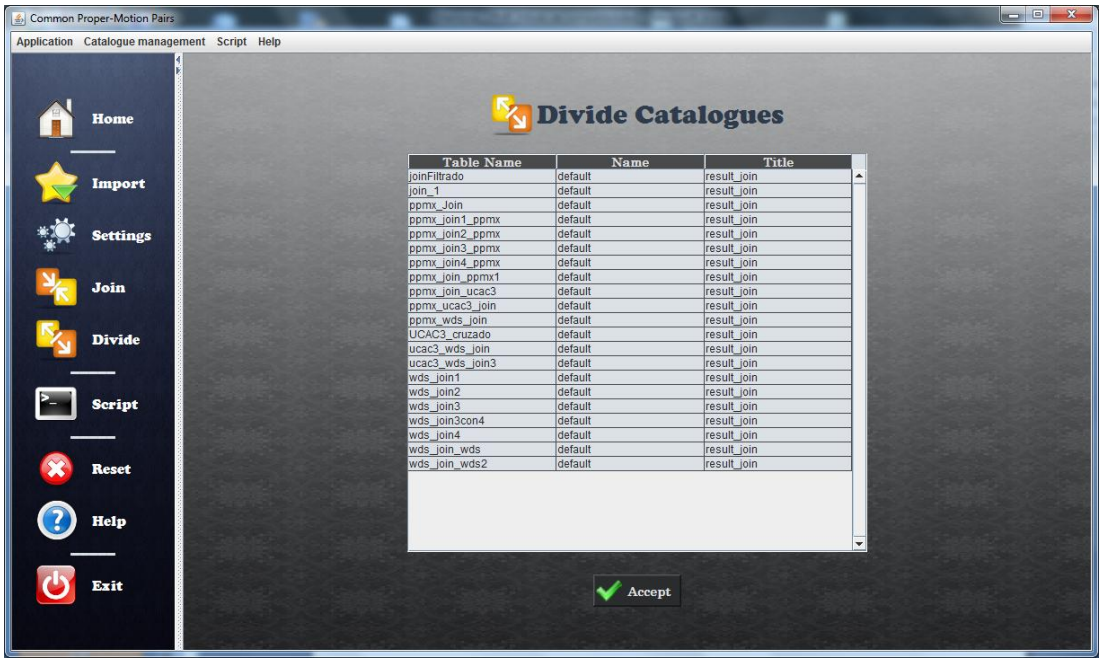


Figura 32. Panel Principal con la opción Divide Catalogues

Para ello el panel muestra una lista de catálogos previamente cruzados (si el usuario todavía no ha cruzado ninguno usando **Join catalogues**, la lista se mostrara vacía) y un botón **Aceptar** para iniciar el proceso. Siguiendo la misma metodología que en los paneles anteriores, basta con seleccionar un catálogo de pares de la lista y pulsar el botón **Aceptar**. Al finalizar la operación se muestra el siguiente mensaje:

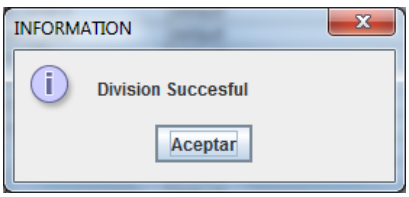


Figura 33. Mensaje de información del Panel Divide

Los dos nuevos catálogos resultados de la división pueden verse listados en el panel **General Settings**, con el mismo nombre que el catálogo de pares origen más un postfijo **_a** y **_b** en cada catálogo.

Script

Para acceder a la ventana que gestiona las funcionalidades relacionadas con los scripts, es necesario pulsar el icono **Script**  o seleccionar el menú **Script** -> **Script Options**  . Una ver realizada dicha acción, se mostrará una nueva ventana dividida en tres sectores (*Figura 34*).

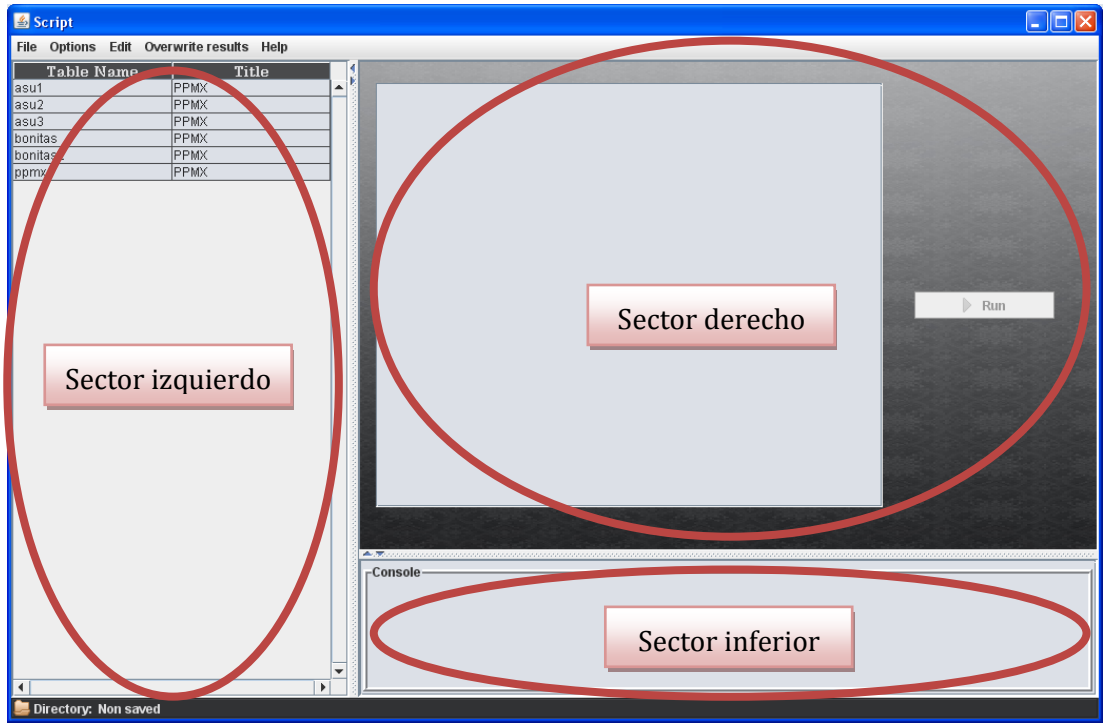


Figura 34. Sectores de la ventana de gestión de scripts (Modo Script)

El sector de la izquierda muestra la lista de catálogos disponibles en la aplicación. A diferencia de la ventana de gestión de catálogos, que mostraba nombre de la tabla, nombre del catálogo de origen, título y número de estrellas, esta lista sólo muestra los nombres de las tablas y su título.

Si se hace doble click sobre un catálogo de la lista, el nombre de este se insertará en el panel de escritura en la posición en la que se encuentre el cursor. Si,

por el contrario, se hace click derecho sobre el catálogo, se podrá visualizar un menú contextual con dos opciones: **Delete** y **Display Fields**, ambos con las funcionalidades descritas en el módulo de gestión de catálogos: el botón **Delete** borrará el catálogo de la aplicación y el botón **Display Fields** mostrará una nueva ventana con los campos del catálogo seleccionado.

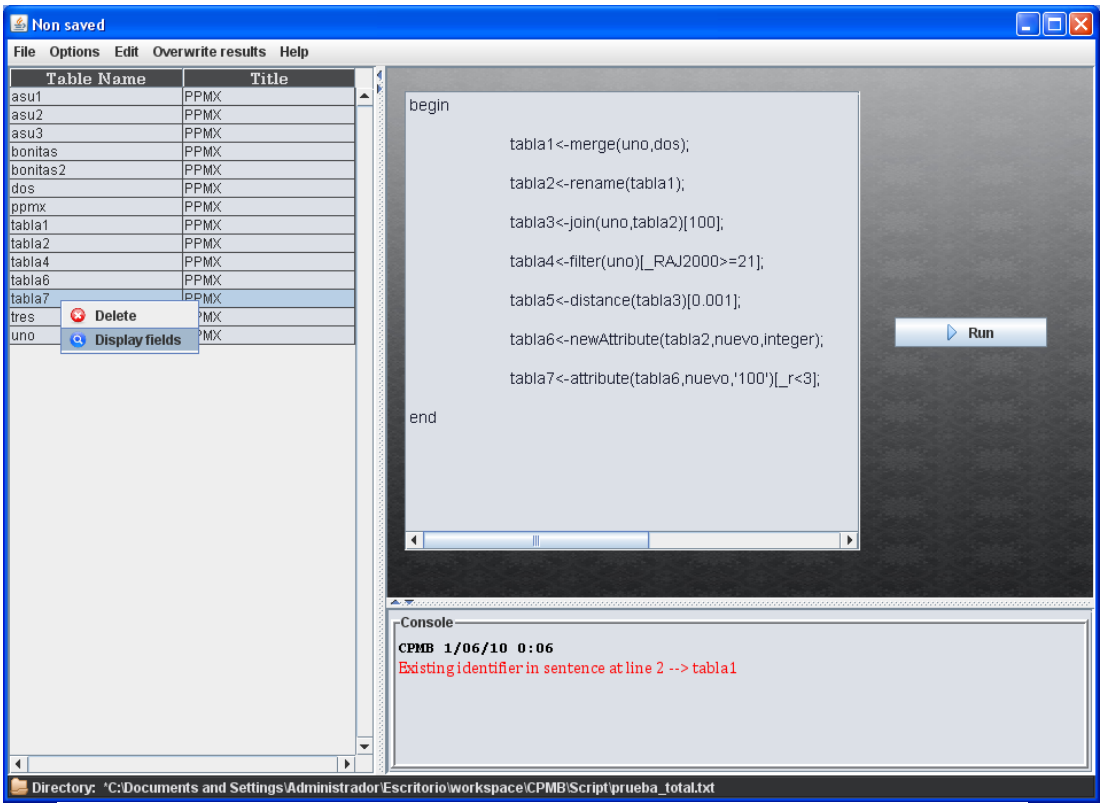


Figura 35. Menú contextual de la lista de catálogos

En el sector de la derecha se observa el panel en el que el usuario podrá escribir sus propios scripts y visualizar los que haya guardado previamente. Junto a dicho panel se encuentra un botón que tiene dos funcionalidades distintas según el modo en el que se encuentre la ventana. Si es el Modo Script, servirá para poder ejecutarlos: botón **Run**. Si, por el contrario, el modo seleccionado es el Modo Función, se usará para poder guardar una función: botón **Save Function**.

Si se hace click con el botón derecho del ratón sobre el panel de escritura, aparecerá un menú de edición contextual, con las opciones **Copy**, **Cut**, **Paste**, **Undo** y **Redo**. Por supuesto, todas estas opciones también están disponibles desde teclado, con las combinaciones de teclas habituales.

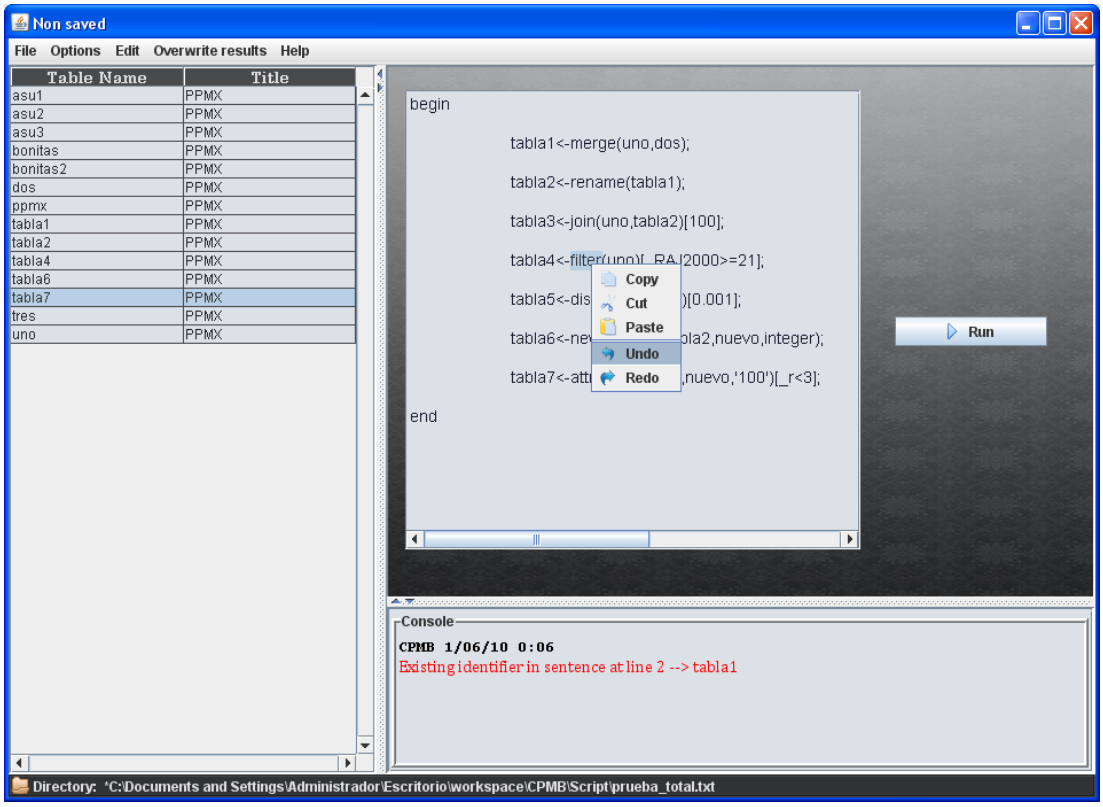


Figura 36. Menú contextual del panel de escritura

En el sector inferior se encuentra la consola en la cual se irán mostrando distintos mensajes al ejecutar un script (errores de compilación y resultados de ejecución de cada una de las instrucciones).

Es posible borrar el contenido de la consola (aunque este es borrado antes de cada ejecución) pulsando con el botón derecho del ratón sobre la misma. Esto provocará que se muestre un menú contextual con una única opción: **Clear Console**.

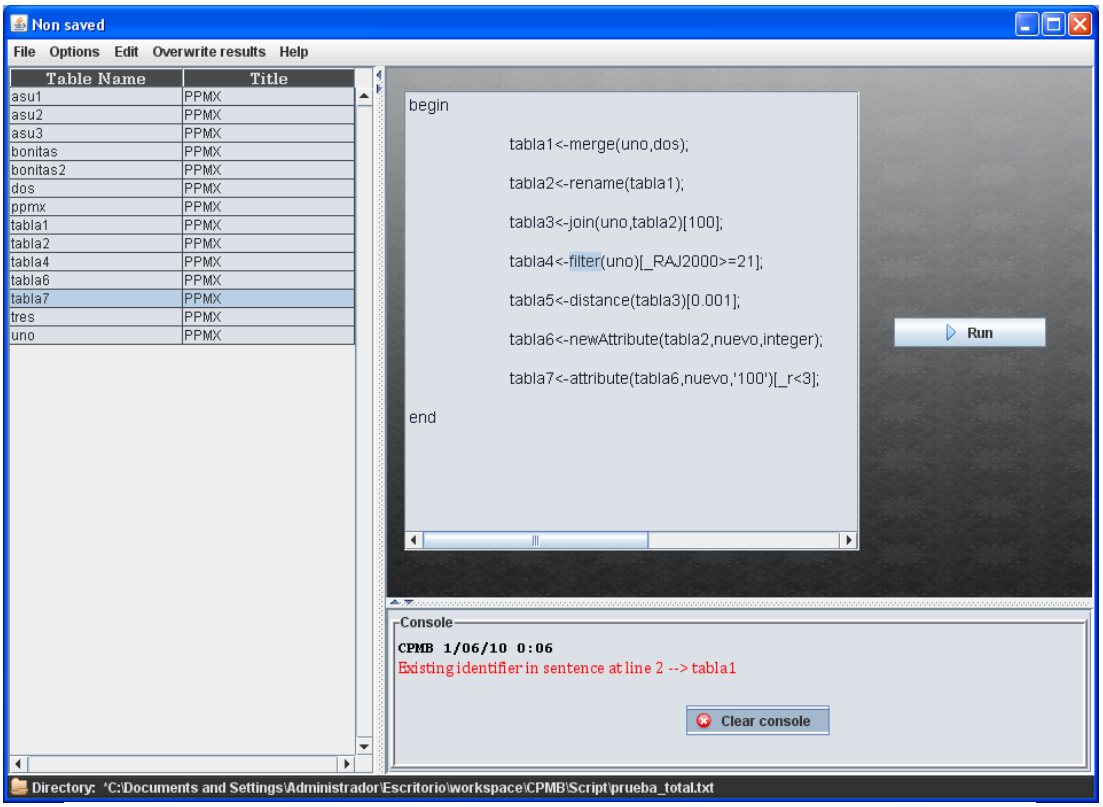


Figura 37. Menú contextual de la consola

En la parte inferior de la ventana se puede observar una barra de estado que muestra la ruta del archivo que se encuentra en el panel de escritura. Por último, en la parte superior, se encuentra una barra de herramientas que nos permitirá ejecutar diversas opciones. Dispone de los siguientes menús y submenús:

- **File**
  - **New**: Limpia el panel de escritura para poder crear un nuevo archivo (script o función, según el modo seleccionado).
  - **Open**: Permite abrir un documento ya guardado (script o función, según el modo seleccionado).
  - **Save** (sólo disponible en el Modo Script): Permite guardar el script que se encuentra modificándose en el panel de escritura.

- **Save as...** (sólo disponible en el Modo Script): Permite guardar el script que se encuentra modificándose en el panel de escritura, proporcionando una nueva ruta.

◊ **Options**

- **Script Mode:** Activa el Modo Script.
- **Function Mode:** Activa el Modo Función.
- **Run** (sólo disponible en el Modo Script): Ejecuta el script que se encuentra en el panel de escritura.

◊ **Edit**

- **Copy:** Copia el texto seleccionado en el portapapeles.
- **Cut:** Corta el texto seleccionado en el portapapeles.
- **Paste:** Pega el texto contenido en el portapapeles en el panel de escritura.
- **Undo:** Deshace la última acción de escritura realizada en el panel.
- **Redo:** Rehace la última acción de escritura, previamente deshecha, en el panel.

◊ **Overwrite results** (sólo disponible en el Modo Script)

- **ON:** Activa la opción de sobreescribir resultados de modo que, al ejecutar instrucciones sucesivas sobre la misma tabla destino, el resultado final de ésta será el de la última instrucción.
- **OFF:** Desactiva la opción de sobreescribir resultados de modo que, al ejecutar instrucciones sucesivas sobre la misma tabla destino, el resultado final de ésta será el de la primera instrucción y las siguientes provocarán un error.

◊ **Help**

- **Help Script:** Abre la ayuda de la aplicación en la sección correspondiente a los scripts.

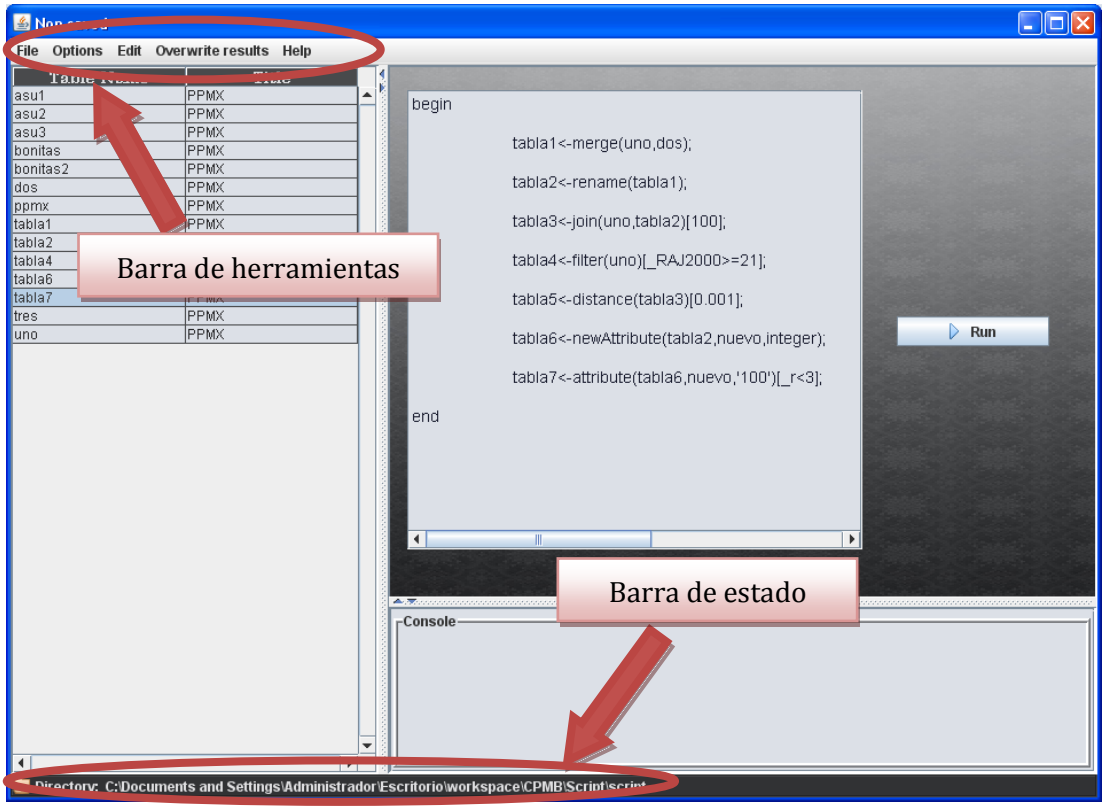


Figura 38. Barra de herramientas y barra de estado

Como se habrá podido deducir, existen dos modos de funcionamiento, Modo Función y Modo Script, que se explican a continuación.

MODO SCRIPT

Este es el modo por defecto de la ventana de gestión de scripts. Para acceder a este modo simplemente hay que seleccionarlo en la barra de herramientas, en el menú **Options**. Desde dicho modo se podrán escribir, ejecutar, abrir o guardar los scripts del usuario (las dos últimas acciones funcionan como en cualquier aplicación que gestione ficheros de entrada/salida).

Escribir un script

Para desarrollar un script lo primero que es necesario conocer es el lenguaje en el que están escritos. Este lenguaje está compuesto por una serie de operaciones básicas que pueden combinarse para obtener el script deseado. Todas las operaciones utilizadas deben acabar con un punto y coma y deben ocupar una

única línea, es decir, no contener ningún salto de línea (si la sentencia es demasiado larga y el panel de escritura la muestra en varias líneas, no habrá problema). Además, dicho conjunto de operaciones debe estar contenido entre las palabras reservadas *Begin* y *End*. Las operaciones básicas del lenguaje, con su sintaxis, se detallan a continuación:

Descripción	Sintaxis
Combinar dos tablas	TablaNueva ← **merge**(tabla1,tabla2)
Renombrar una tabla	TablaNueva ← **rename**(tabla)
Producto cartesiano de dos tablas obteniendo una tabla de pares	TablaNueva ← **join**(tabla1, tabla2)[segundos]
Filtrar una tabla eliminando las filas que no cumplan una condición dada	TablaNueva ←**filter**(tabla)[condicionSQL]
Eliminar de una tabla de pares las parejas que están a más de una distancia dada	TablaNueva ←**distance**(tabla)[segundos]
Añadir un campo nuevo a una tabla	TablaNueva ← **newAttribute**(tabla, AtributeName, AttributeType)
Dar valor a un campo en todas las filas que cumplan una condición dada	TablaNueva ← **attribute**(AttributeName, Value)[condición SQL]
Eliminar un campo	TablaNueva ← **deleteAttribute**(tabla , AttributeName)
Obtener una nueva tabla con solo aquellas filas de la tabla1 que no estén en la tabla2	TablaNueva ← **minus**(Tabla1,Tabla2)

Un ejemplo de script puede verse en el siguiente cuadro de texto (la sentencia *calculaMovPropio(temp1 , tempAux);* será comentada más adelante):

```
begin
        temp1<- join(ppmX, ppmX)[10];
         calculaMovPropio(temp1 , tempAux);
        temp2<- filter(tempAux)[mu_A>= 60 and mu_B >= 60];
        temp3<-filter(temp2)[Vmag is not null  ];
        tablaResultado<-minus(temp3, wds);
end
```

Ejecutar un script

Para poder ejecutar un script es necesario que éste se encuentre visible en el panel de escritura. Una vez escrito o cargado el script, será necesario pulsar el botón **Run** que se encuentra a la derecha del panel (también es posible ejecutar el script seleccionando la opción **Run** en el menú **Options** de la barra de herramientas). En este momento comenzará la compilación y ejecución de cada una de las instrucciones que componen el script. Si alguna de ellas tiene errores de sintaxis, los parámetros son erróneos o sucede algún error durante la ejecución, el usuario será advertido mediante mensajes en color rojo que se mostrarán en la consola (*Figura 39*). Si, por el contrario, la ejecución finaliza correctamente, la consola mostrará una línea en color azul con el mensaje "Compiled Succesfully" (*Figura 40*). Además de estos mensajes, el resultado de la ejecución individual de las instrucciones será mostrado al finalizar cada una de estas, junto con el valor de la opción **Sobreescribir resultados**.

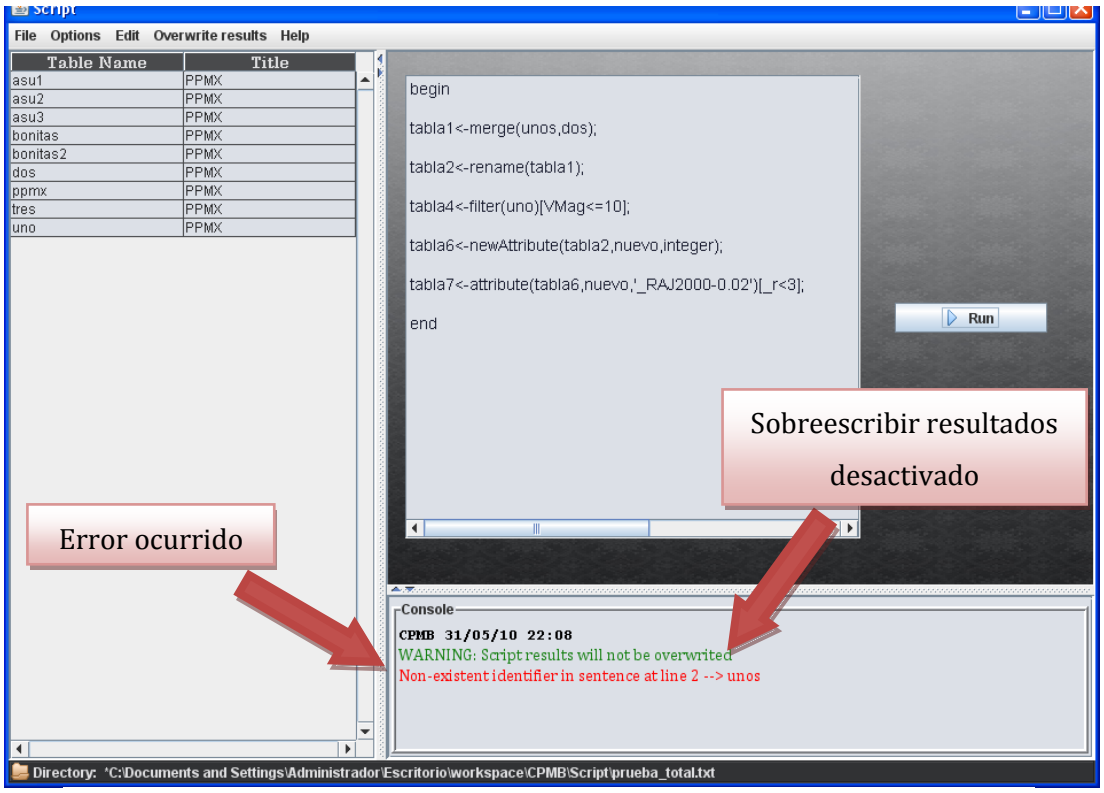


Figura 39. Ejecución con errores

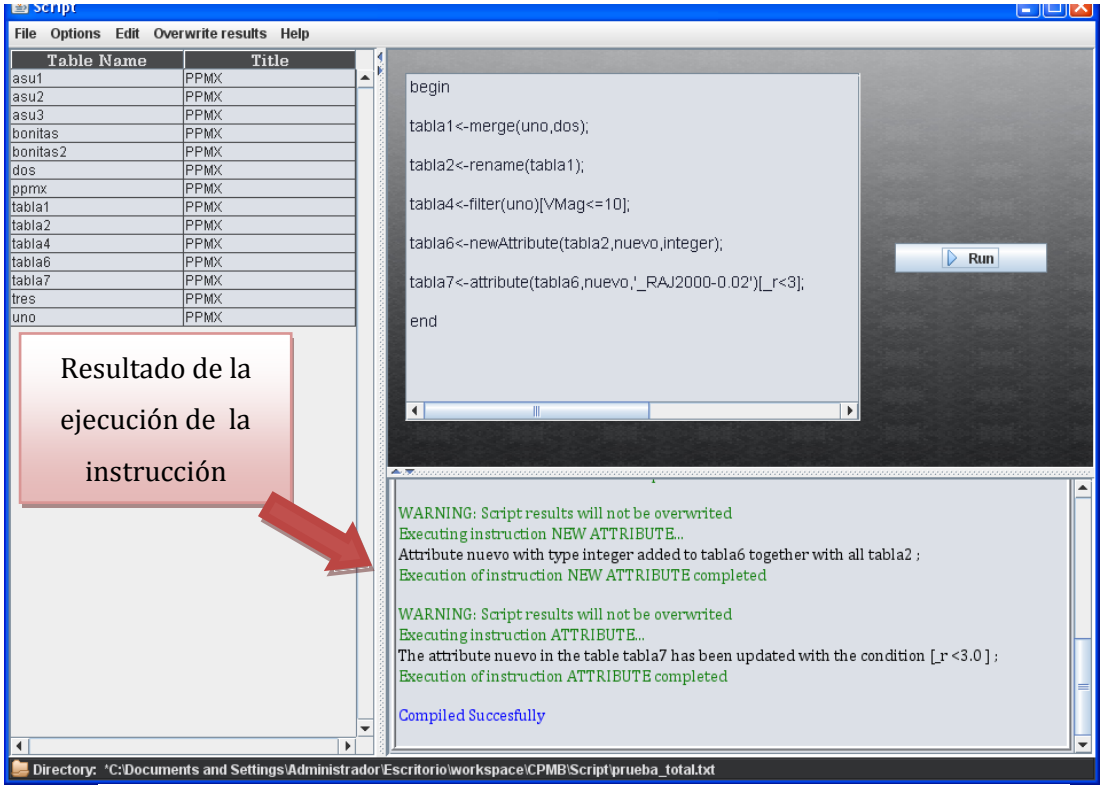


Figura 40. Ejecución correcta

MODO FUNCIÓN

Este es el modo en el que la aplicación permite al usuario escribir sus propias funciones parametrizadas. Dichas funciones pueden ser muy útiles para definir las operaciones más comunes, de forma que puedan ser usadas en diferentes scripts, o crear otras nuevas mucho más complejas a partir del repertorio básico ofrecido al usuario.

Para acceder a este modo simplemente hay que seleccionarlo en la barra de herramientas, en el menú **Options**. Una vez realizada esta acción, se pueden observar algunos cambios con respecto al Modo Script: en la barra de herramientas, el menú **File** sólo contiene dos submenús (**Open** y **Save**); el menú **Options** sólo contiene los submenús correspondientes al cambio de modo; el menú **Overwrite Results** ya no es visible y el botón a la derecha del panel de escritura ha cambiado su nombre por el de **Save Function**. Todos estos cambios son debidos a que la única misión de las funciones es ser guardadas, y no ejecutadas, para su posterior uso en algún script.

La sintaxis de las funciones es similar a la de los script, con una salvedad: las funciones deben comenzar una cabecera que empezará con la palabra reservada *Function* y el nombre de la función. A continuación debe ir el número de parámetros entre paréntesis.

El cuerpo de la función es igual que el de un script, sólo que los nombres de los catálogos deben ser sustituidos por **$i**, siendo **i** el número que identifica al parámetro en la llamada a la función. El siguiente ejemplo esclarece esta explicación:

```
Function calculaMovPropio(2)
begin
  aux1<- newAttribute($1,mu_A,double);
  aux2<- newAttribute(aux1,mu_B,double);
  aux3<-attribute(aux2, mu_A, 'sqrt(pmra * pmra + pmde * pmde)' );
  $2<-attribute(aux3, mu_B, 'sqrt(b_pmra * b_pmra + b_pmde * b_pmde)' );
end
```

En la definición de la función *calculaMovPropio* se puede apreciar que tiene dos parámetros, $1 y $2. Estos parámetros pueden ser de entrada o salida, pero no se puede distinguir cuál es cada uno únicamente con la cabecera de la función: es necesario observar el cuerpo de la misma. De las instrucciones 1 y 4 se deduce que $1 actúa como parámetro de entrada y $2, de salida. Es muy importante conocer cuál es el tipo de cada uno de los parámetros, puesto que ese será el modo en el que deberán ser especificados en la llamada a la función. Un cambio de orden de los parámetros en la llamada puede dar lugar a errores de ejecución o a la sobreescritura de tablas con valores erróneos.

Una vez escrita la función, es necesario almacenarla correctamente para que pueda ser utilizada en los scripts. Para ello debe ser guardada en la carpeta **Library**, que se encuentra en el mismo directorio que la aplicación. Además el archivo de la función debe ser guardado con el mismo nombre que ésta y con extensión **.f**, de lo contrario, no podrá ser invocada por ningún script. Para ello se debe pulsar el botón **Save Function**  , que se encuentra a la derecha del panel de escritura. Realizados estos pasos, es posible realizar la llamada a la función desde un script:

```
begin
        temp1<- join(ppmX, ppmX)[10];
         calculaMovPropio(temp1 , tempAux);
        temp2<- filter(tempAux)[mu_A>= 60 and mu_B >= 60];
        temp3<-filter(temp2)[Vmag is not null  ];
        tablaResultado<-minus(temp3, wds);
end
```

En el ejemplo se observa que la llamada a la función *calculaMovPropio* se realiza con los parámetros *temp1* y *tempAux*. Como se ha explicado antes, no es fácil conocer a priori el tipo de los parámetros sin conocer la definición de la función. En este caso, observando dicha definición, se advierte que *temp1* actúa como parámetro de entrada y *tempAux* como parámetro de salida.

APÉNDICE 2: DESCARGAR CATÁLOGOS DE VIZIER

VizieR permite obtener información de todos los catálogos disponibles de forma gratuita. En este caso los catálogos más interesantes son:

- NOMAD: Incluye datos de velocidades (movimientos propios) y magnitudes.
- TYCHO: Incluye datos de velocidades (movimientos propios) y magnitudes.
- GSC 2.3.2: Sólo incluye magnitudes.
- PPMX: Incluye datos de velocidades y magnitudes.
- UCAC3: Sólo incluye datos de velocidades.
- WDS: Catálogo de estrellas dobles ya conocidas.

En este pequeño apéndice se explicará cómo descargar un intervalo de estrellas incluidas en el catálogo PPMX *(Position and Proper Motions eXtended)* en un fichero de texto valido para importar a CPMP.

Lo primero de todo es abrir el portal de VizieR en el navegador (http://vizier.u-strasbg.fr/viz-bin/VizieR).

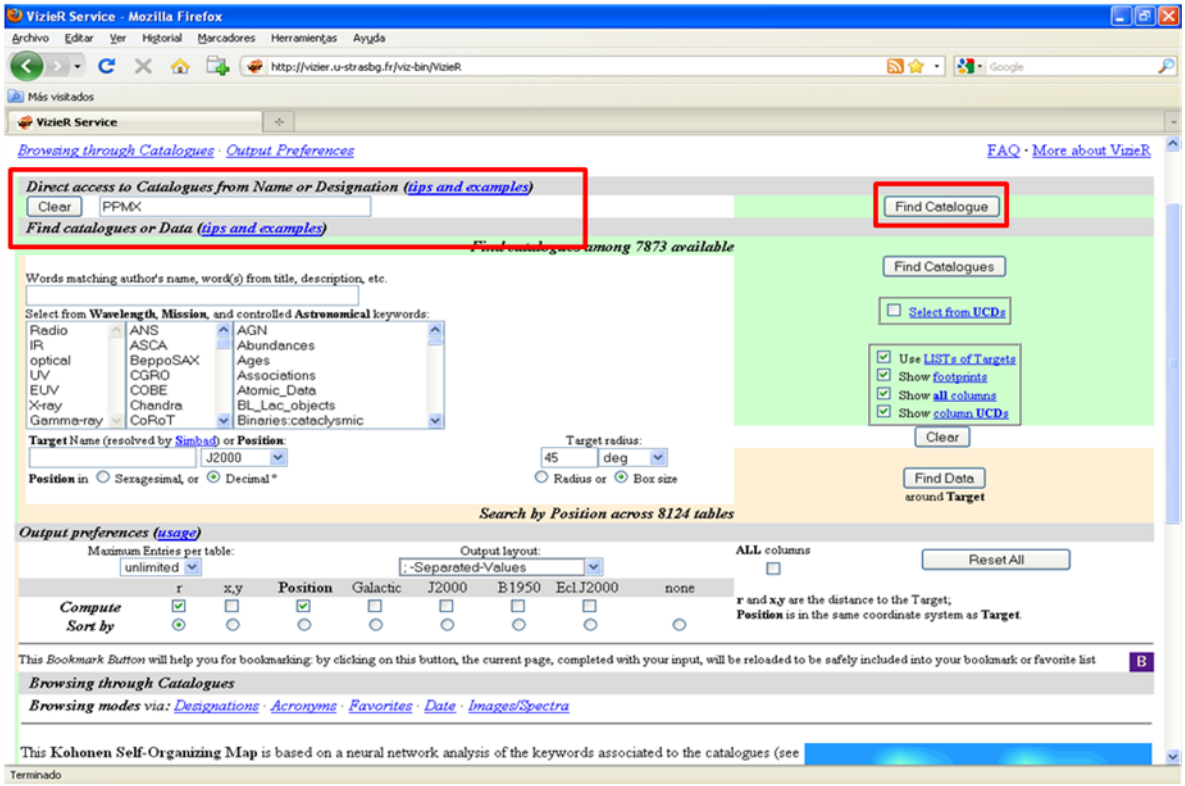


Figura 1. Página web de Vizier

En la página web se observan diversas opciones que completar. Para el ejemplo concreto que se propone, se debe rellenar el área de texto especificado como ***Direct access to Catalogues from Name or Designation*** con el nombre del catálogo elegido, "PPMX", y después pulsar el botón ***Find Catalogue***. (*Recuadrados en rojo en la Figura 1*). Esta acción actualizara la página web a la siguiente:

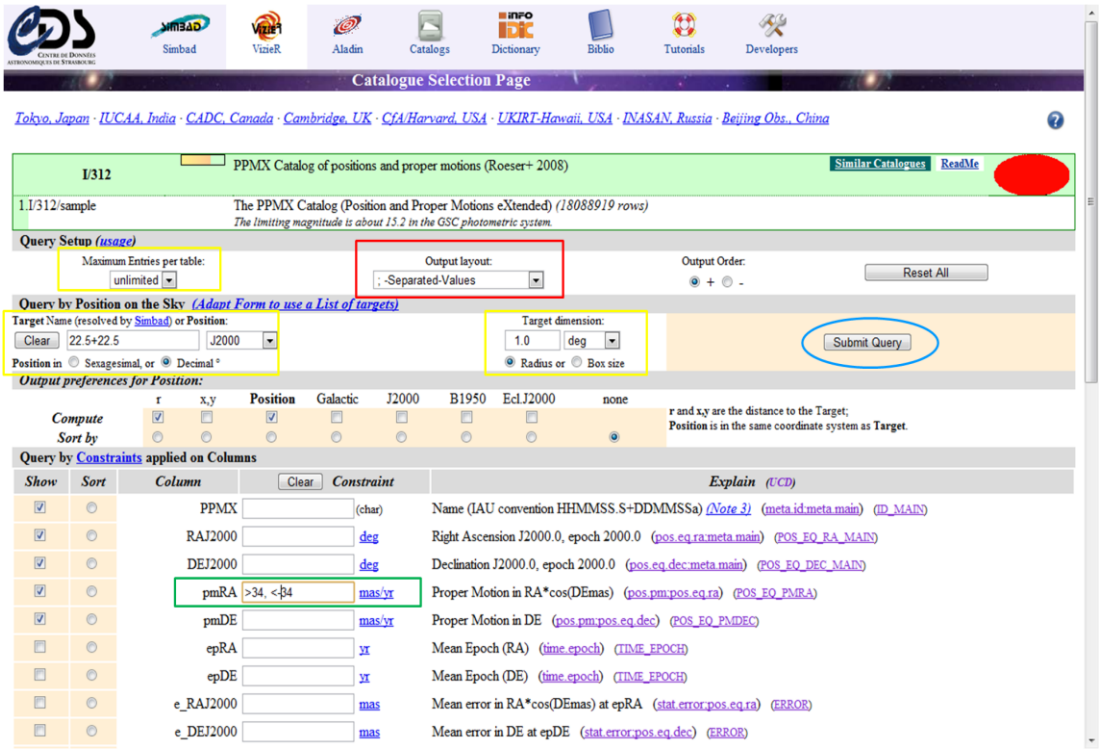


Figura 2. Página web de Vizier seleccionado el catálogo PPMX

En ella se vuelve a observar muchas opciones de selección distintas. La más importante para que el catálogo descargado tenga el formato requerido por CPMP es la opción ***"; -Separated-Values"*** , que se encuentra en la lista de selección ***Ouput Layout*** (*recuadro rojo en la Figura 2*).

Algunas de las demás opciones serán explicadas para un ejemplo concreto, como puede ser descargar del catálogo PPMX un intervalo de radio 1.0 grado a partir de la coordenada 22.5 + 22.5 (AR+ DE) en formato decimal.

Para especificar la coordenada sobre la que se desea realizar la consulta es necesario rellenar el campo de texto ***Target Name or Position***. El intervalo de la consulta será introducido rellenando la opción ***Target Dimension*** con 1.0 ***deg*** y ***Radius***. Si además se desea asegurar la inclusión en el fichero resultado de todas

las filas consultadas es necesario elegir **unlimited** en la lista de selección de ***Maximum Entries per table*** (*Recuadros amarillos en la Figura 2*).

Otras opciones que también pueden resultar interesantes son las restricciones en las características de las estrellas a descargar. Por ejemplo, seleccionar solo las estrellas con pmRA mayor a 34 mas/year (*Proper Motion in RA*cos(DEmas)*) (*Opción recuadrada en verde en la Figura 2*).Los valores pmRA y pmDE son la velocidad en milisegundos de arco por años. En este caso puede que sólo interesen las estrellas con movimiento elevado, ya que a partir de movimientos elevados y similares en un par de estrellas se podrán descubrir estrellas que viajan "juntas", es decir, pares con movimiento propio común.

Al finalizar la selección de las características deseadas se pulsará el botón **Submit Query** (*elipse azul en la Figura 2*) y se mostrará un cuadro de dialogo para que se seleccione el directorio donde se desea descargar el archivo con los datos seleccionados.

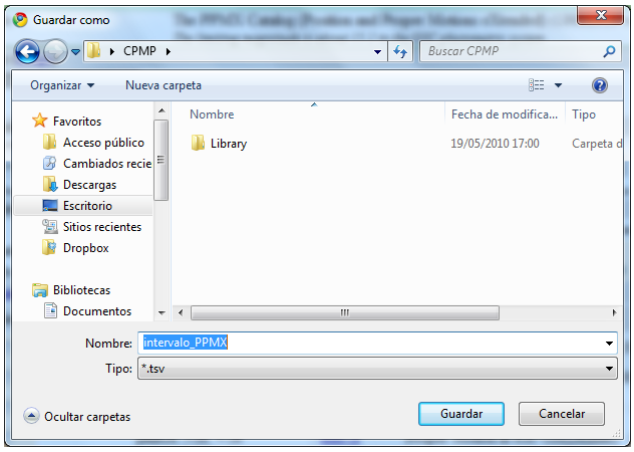


Figura 3. Cuadro de diálogo de selección del directorio destino

Una vez descargado ya se dispone de un primer catálogo que importar en CPMP (*ver Apéndice 1: Manual, sección Gestión de Catálogos*).

Es posible encontrar más tutoriales sobre VizieR en http://aladin.u-strasbg.fr/tutorials/vizier.gml

Vol. 6 No. 3 July 1, 2010 Journal of Double Star Observations Page 206

New Common Proper-Motion Pairs from the PPMX Catalog

Rafael Caballero, Blanca Collado- Iglesias, Sara Pozuelo- González,
Antonio Fernández- Sánchez

Agrupación Astronómica Hubble,
Martos, Jaén, Spain

Email: rafa@sip.ucm.es

Abstract: We use data mining techniques for finding 82 previously unreported common proper motion pairs from the PPM-Extended catalogue. Special-purpose software automating the different phases of the process has been developed. The software simplifies the detection of the new pairs by integrating a set of basic operations over catalogues. The operations can be combined by the user in scripts representing different filtering criteria. This procedure facilitates testing the software and employing the same scripts for different projects.

Introduction

In previous papers (Caballero 2009, Caballero 2010) we data mined different catalogs using some criteria to obtain new common proper-motion pairs (CPMP's from now on) not included in the WDS (Washington Double Star Catalog, Mason, et al., 2003). This idea is not new and has been used for instance by Greaves (2004).

During the development of the projects it became clear that the process was almost the same in all the cases, with a few changes due to the particular characteristics of each catalog. Therefore it seemed interesting to develop a special-purpose software. Such a project was suggested as a Master's thesis topic at the faculty of Computer Science at the University Complutense of Madrid (Spain). The project was developed by Blanca Collado-Iglesias, Sara Pozuelo-González and Antonio Fernández-Sánchez, and directed by Rafael Caballero. This paper presents 82 new CPMPs from the PPM-Extended catalog (PPMX, see Röser, 2008) obtained with the help of this application.

The Data Mining Process

The overall data mining process can be described as follows:

1. Downloading (part of) the main catalog *C* , usually from the online VizieR Service web page (Allende & Dambert 1999). Sometimes portions of auxiliary catalogs are also needed, for instance to complete the information about spectra, visual magnitude, etc.
2. Importing *C* data into a relational database such as Access or MySQL (and also the auxiliary catalogs).
3. Obtain the Cartesian product $D = C \times C$. *D* is thus a table of pairs.
4. Delete from D all the pairs with separation greater than some arbitrary number, for instance 100 seconds.
5. Remove from D all the pairs that are already part of the WDS.
6. Apply some criteria, in order to keep in D only possible CPMPs. A typical case is the Halbwachs' criteria (Halbwachs, 1986).
7. If possible, introduce further criteria that can help to increase the data quality, i.e. to reject those pairs that are more likely not physically attached.
8. Finally, check every pair in the photographic plates available at ALADIN (Bonnarel et al., 2000), looking for two stars with noticeable motion and roughly the same astrometry data in the expected position.

9. Complete the data with astrometry and other suitable data from auxiliary catalogs.

The Software Application

The opening screen of the application is shown in Figure 1. It was developed in the Java programming language. This language was chosen because it easily allows connecting to different databases using a convenient JDBC (Java Database Connectivity) driver. Initially, the system is configured for using the relational database MySQL, but it can be readily adapted for other databases such as Oracle or Access. The program allows the user importing catalogs obtained from Vizier in text format with the fields separated by ";". By parsing the header produced by VizieR, the type and size of the different attributes is detected, and a suitable SQL table created.

Other options for general management of catalogs are included. Probably one of the most useful ones is the "Join Catalogs" option. With this option the user can cross two tables containing individual stars yielding a new table containing those pairs with separation less than a parameter in seconds (steps 3 and 4 of the overall process described in the previous section). This table will contain the set of initial candidates.

Figure 1: Application main screen.

However, the most appealing feature of the program is that it allows the users to program their own scripts for filtering the candidate pairs. The scripts can be saved and are defined by combining a few core operations, like adding a new field to an existing table, setting the value for some field for all the rows fulfilling some condition, combining catalogs to produce a new one, or deleting all the rows that do not satisfy a given condition. Furthermore, the system allows defining parameterized functions that will be used from different scripts. As a simple example consider the following user-defined function:

```
function setFilterPM(2)
begin
 temp1<-newAttribute($1,mu,double);
 temp2<-newAttribute(temp1,b_mu,double);
 temp3<-attribute(temp2,mu,'sqrt
(pmra*pmra+pmde*pmde)')[true];
 temp4<-attribute(temp3,b_mu,'sqrt
(b_pmra*b_pmra+b_pmde*b_pmde)')[true];
 $2<-filter(temp4)[mu>=50 and b_mu>=50];
end
```

The function receives as input a table $1 of pairs, which is assumed to contain fields *pmra*, *pmde*, *b_pmra*, and *b_pmde*, which represent the proper motion in RA and DEC of the two components, and it creates a new table $2 such that:

- It contains two new fields *mu* and *b_mu* such that for each row $mu = \sqrt{(pmra^2 + pmde^2)}$ and $b\_mu = \sqrt{(b\_pmra^2 + b\_pmde^2)}$.
- It only contains those rows verifying *mu*>=50 and *b_mu*>=50 (i.e. both components with proper motion over 50 millisecond of arc/year).

The function is self-explanatory: the two first statements after the reserved word *begin* add the new fields, both of type *double* (real numbers). The two next lines give value to the new fields. The condition *[true]* at the end of each statement specifies that the new values must affect all the rows. Finally, the last statement before the reserved word *end* removes all the rows corresponding to pairs where some of the components have proper motion below 50 mas/yr.

A Test Case: the PPMX Catalog

In order to check the software the PPMX catalog was chosen. In particular we started downloading the catalog for entries with available V magnitude and with proper motion over 50 milliseconds of arc per

year. Then we used the option "join catalogs" to produce the initial set of candidate pairs. 2171 pairs with separation under 100 seconds were obtained. Then a new, more restrictive, version of the Halbwachs' criteria was applied (see next section), further reducing the set of pairs to 979. Then the pairs already in the WDS, or those likely to be included in the catalog in the near future, were removed leaving 85 pairs. Another three pairs were excluded after using the Reduced Proper Motion (RPM) discriminator proposed by Salim & Gould (2003). Finally, all the pairs were checked in the photographic plates, finding all of them. The summary of this process is shown in Table 1.

Table 1: Summary of the data mining process for PPMX

Phase	Rows
PPMX initial subset	172115
Candidate pairs	2171
After (modified) Halbwachs criteria	979
After removing pairs already in WDS	213
After removing pairs in other lists	85
After RPM criterion	82
After checking photographic plates	82

Halbwachs revisited

The three criteria originally proposed by Halbwachs for distinguishing physical and optical pairs from their proper motion are:

1. $(\mu_1 - \mu_2)^2 < -2\,(\sigma_1^2 + \sigma_2^2)\ln(0.05)$
2. $|\mu_1|, |\mu_2| \geq 50$ mas/yr
3. $\rho / |\mu_1|, \rho / |\mu_2| < 1000$ yr

where μ_1, μ_2 are the two proper motion vectors, σ_i is the mean error of the projections on the coordinate axes of μ_i, and ρ is the angular separation of the two stars. The first condition checks if the hypothesis $\mu_1 = \mu_2$ is admissible with a 95% confidence considering the given errors σ_1 and σ_2. If $\mu_1 = (\mu_{11}, \mu_{12})$, $\mu_2 = (\mu_{21}, \mu_{22})$ then this condition can be rewritten as

(1') $(\mu_{11} - \mu_{21})^2 + (\mu_{21} - \mu_{21})^2 < -2\,(\sigma_{12} + \sigma_{22})\ln(0.05)$

However in previous experiences it was observed that this criterion allowed pairs with noticeably different values in some axes, and thus some additional criterion was needed. In this project we propose using the condition for each axis separately, i.e. to replace condition (1) (or (1')) by:

(1.a) $(\mu_{11} - \mu_{21})^2 < -2\,\sigma_{12}\ln(0.05)$
(1.b) $(\mu_{12} - \mu_{22})^2 < -2\,\sigma_{22}\ln(0.05)$

It is straightforward to check that the conjunction of (1.a) and (1.b) imply (1'), and therefore the new condition is more restrictive. In particular, in the case of the PPMX project replacing the condition (1) by (1.a) and (1.b), results in 15 additional pairs filtered out. These pairs are precisely those with noticeable differences in any axis. Hence, we think that substituting (1) by (1.a) and (1.b) is a good practice that improves the quality of the results.

Results

Table 2 shows the final list with the new CPMP obtained. The astrometry (precise coordinates, position angle, separation, and date) have been obtained from the 2MASS catalog. The visual magnitudes correspond to the values in PPMX. The pair separation ranges from 8.64" to 99.40", and the visual magnitudes are between 6.25 and 13.07. The pair at RA-dec 21 36 58.08 -35 53 02.93 is especially remarkable, since the parallax of both components is known and compatible: 14.95 ±0.93 mas for A and 15.09 ±1.21 mas for B. In this case it seems quite safe to say that this bright pair (mags. 7.35 and 8.60) is physically attached. For another 15 pairs the parallax of the primary is known, and for a larger number the spectral type of one or the two components is available either from Hipparcos (Perryman, 1997) or from Tycho-2 (Wright, 2003). In three cases the secondary of the new pair is a close pair already in WDS, see notes 13, 21, and 34.

Table 3 contains the proper motion data of the new CPMP. The data corresponds to PPMX.

Conclusions

Ensuring the data quality must be one of the main goals of any data mining project. With this purpose we have developed a software application that simplifies the different phases of the project. The filters for selecting and reducing the number of candidate pairs are easily prepared by the user by employing a small subset of basic operations over catalogues, which constitutes the core language offered by the application. The

(Continued on page 215)

New Common Proper-Motion Pairs from the PPMX Catalog

Discovery Designation	RA DEC	Mags		PA	SEP	DATE	NOTES
CBL	00 32 19.099 -21 50 33.69	11.7	12.29	208.45	49.78	2000.783	
CBL	00 55 33.213 -43 16 11.73	12.24	12.63	134.53	28.20	1999.713	
CBL	01 00 52.512 -18 56 57.08	11.44	11.57	220.88	25.25	1998.626	
CBL	01 52 27.571 -49 31 25.56	9.04	9.16	207.56	42.41	1999.812	(1)
CBL	02 04 18.761 -70 59 40.88	10.48	11.77	103.88	24.05	1999.894	
CBL	02 05 27.334 +38 20 57.02	12.29	13.07	130.16	31.88	1998.810	
CBL	03 10 41.550 -20 06 41.54	7.62	10.61	313.69	59.79	1998.878	(2)
CBL	03 24 54.681 -43 12 55.77	9.86	11.62	192.09	26.63	1999.648	(3)
CBL	03 46 09.569 -41 12 22.33	9.25	11.48	257.08	66.28	1999.629	(4)
CBL	04 23 48.192 -76 43 09.45	11.13	11.29	265.20	45.21	1998.840	
CBL	05 01 36.173 -44 49 49.05	7.55	10.58	265.53	50.77	1999.722	(5)
CBL	05 23 39.978 -38 18 48.09	11.82	12.20	193.43	24.06	1999.173	
CBL	05 57 24.758 -40 23 51.78	8.15	11.28	350.10	45.10	1999.190	(6)
CBL	06 15 01.664 -63 20 38.82	11.71	12.57	219.59	15.59	1998.947	
CBL	06 38 17.669 +18 28 24.58	11.44	11.79	63.63	55.81	1997.898	
CBL	07 03 08.446 -73 50 13.91	11.12	11.34	140.15	51.35	2000.167	
CBL	07 08 55.244 -11 23 29.58	11.22	12.32	31.21	48.15	1999.138	
CBL	09 33 19.911 -07 11 24.75	6.25	10.78	19.14	57.81	1999.048	(7)
CBL	09 51 08.004 -18 39 31.43	7.37	10.83	115.95	50.68	1998.322	(8)
CBL	10 14 38.104 -13 33 29.10	9.04	10.34	177.99	28.62	1999.299	(9)
CBL	10 15 08.028 -65 26 11.04	11.15	11.47	304.57	31.11	2000.230	
CBL	10 16 15.352 -17 11 13.12	10.67	11.78	282.61	41.57	1998.234	
CBL	10 32 03.297 -30 28 05.49	11.46	11.78	29.56	14.49	1999.223	
CBL	10 47 27.741 -11 54 08.72	9.83	10.46	48.64	27.76	1998.256	(10)
CBL	11 29 03.259 -38 17 05.77	9.48	11.35	109.61	38.72	1999.272	(11)
CBL	11 35 52.047 -40 40 36.43	11.85	12.18	247.31	20.08	1999.275	
CBL	11 53 22.088 -67 07 05.56	10.78	11.88	119.24	36.11	2001.121	
CBL	12 05 25.156 +17 17 21.69	7.63	10.04	311.55	43.20	1998.033	(12)
CBL	12 13 30.463 -48 47 46.64	8.80	10.57	347.83	49.99	1999.357	(13), (14)
CBL	12 34 19.535 -35 22 46.48	10.46	11.63	162.81	44.30	1999.258	
CBL	12 35 15.748 -09 10 57.84	10.91	10.95	142.98	28.20	1999.089	

Table continues on next page.

New Common Proper-Motion Pairs from the PPMX Catalog

Discovery Designation	RA DEC	Mags		PA	SEP	DATE	NOTES
CBL	12 35 43.067 -03 00 58.10	8.49	9.79	280.18	60.70	1999.149	(15)
CBL	12 36 16.407 -79 31 34.45	10.95	11.34	254.57	14.99	2000.102	
CBL	13 17 35.429 -11 57 01.34	11.04	12.70	344.71	24.29	1999.138	
CBL	13 53 54.390 -07 45 44.87	11.30	11.84	216.02	18.76	1999.163	
CBL	14 08 01.294 -13 16 09.22	11.84	12.31	180.53	13.24	1999.299	
CBL	14 12 31.776 -30 06 17.68	11.48	11.77	216.48	31.32	1999.262	
CBL	14 35 32.025 -35 26 39.17	11.44	11.72	211.37	41.19	2000.310	
CBL	14 37 23.205 -66 50 27.83	9.95	10.47	305.18	27.02	2000.258	(16)
CBL	14 53 52.152 -63 53 53.17	9.28	11.02	83.26	41.56	2000.223	(17)
CBL	14 55 28.254 -56 48 55.15	9.85	11.00	265.59	26.29	2000.146	(18)
CBL	14 58 31.045 -27 24 06.18	10.79	11.48	91.00	15.09	1998.486	
CBL	15 04 08.091 -26 23 26.83	10.97	11.56	322.81	26.24	1998.486	
CBL	15 07 12.834 -41 41 31.12	9.36	9.46	286.51	8.64	1999.374	(19)
CBL	15 13 59.433 -58 37 15.68	9.88	10.30	319.51	45.32	1999.431	(20)
CBL	16 31 42.851 +70 55 59.84	8.22	11.45	312.31	39.95	1999.398	(21), (22)
CBL	16 37 35.305 +69 19 17.25	9.07	10.76	124.95	99.40	1999.399	
CBL	17 01 49.674 +14 42 27.70	10.37	10.63	12.39	19.29	1999.158	
CBL	17 40 21.442 +05 43 37.44	11.19	11.98	342.16	34.69	2000.404	
CBL	17 54 51.203 +28 51 38.93	11.27	12.36	244.01	41.01	2000.204	
CBL	17 56 59.673 -46 06 31.92	10.91	10.94	207.38	11.21	1999.551	
CBL	17 59 53.975 -45 17 20.72	10.94	11.67	192.86	14.26	1999.551	
CBL	18 07 28.944 +00 29 27.19	11.23	11.54	355.05	49.82	1999.548	
CBL	18 13 05.162 +18 40 45.60	8.37	10.50	260.54	34.30	2000.209	(23)
CBL	18 21 26.185 -15 22 18.08	9.72	11.08	26.29	18.82	1999.333	
CBL	18 22 44.038 -40 44 59.30	10.32	10.73	159.99	12.69	2000.427	
CBL	18 27 24.762 +21 51 53.42	10.22	11.99	159.94	26.67	2000.242	
CBL	18 38 36.531 +49 00 42.13	9.42	10.72	260.52	47.47	1998.478	(24)
CBL	18 41 25.436 -44 32 30.63	10.78	11.14	53.29	36.30	2000.474	
CBL	18 45 04.604 -23 15 07.22	8.45	11.24	93.96	25.15	1998.478	(25)
CBL	18 54 43.138 -50 07 46.85	9.18	11.72	280.23	27.51	1999.726	(26)
CBL	19 01 33.281 -24 08 28.08	9.20	11.43	83.49	31.00	1999.262	(27)

Table concludes on next page.

New Common Proper-Motion Pairs from the PPMX Catalog

Discovery Designation	RA DEC	Mags		PA	SEP	DATE	NOTES
CBL	19 07 06.084 -14 04 09.97	8.64	10.38	317.61	20.69	2000.247	(28)
CBL	20 06 03.948 -41 37 36.45	11.49	11.68	8.99	23.71	1999.505	
CBL	20 33 53.250 -27 10 17.31	9.35	11.97	39.94	52.00	2000.561	(29)
CBL	20 36 05.730 -67 05 22.53	10.50	11.32	107.53	24.53	2000.542	
CBL	20 46 51.514 -49 28 39.62	10.71	11.39	197.77	27.90	1999.710	
CBL	21 10 18.359 -13 04 05.84	11.13	11.15	246.88	16.80	1999.483	
CBL	21 16 13.422 -40 40 51.94	11.30	12.07	151.91	31.23	1999.691	
CBL	21 29 36.229 -44 13 50.10	10.36	10.71	261.71	90.54	1999.633	
CBL	21 36 58.092 -35 53 03.02	7.35	8.60	265.34	78.45	2000.562	(30)
CBL	21 54 22.594 -44 09 46.37	11.10	11.90	73.80	17.96	1999.718	
CBL	22 08 27.535 -57 06 52.51	11.08	11.45	327.83	26.33	2000.543	
CBL	22 09 42.632 -33 45 15.41	9.28	10.25	278.32	49.90	1999.560	(31)
CBL	22 32 09.402 -13 35 51.81	7.72	9.71	93.97	41.94	1998.481	(32)
CBL	22 33 45.700 +61 45 26.85	9.94	10.90	222.22	38.29	1999.746	
CBL	22 41 49.606 +59 47 35.64	9.03	11.02	104.02	47.92	1999.741	(33)
CBL	22 47 55.497 +03 36 07.26	11.28	11.35	154.24	23.20	2000.608	
CBL	22 53 55.679 -37 09 40.50	10.22	10.59	313.30	54.44	1999.734	
CBL	22 54 19.523 +30 22 18.31	10.46	11.40	233.61	14.46	1998.473	
CBL	23 28 08.467 -02 26 53.36	8.05	8.43	226.37	57.33	1998.730	(34), (35)
CBL	23 37 40.086 +00 46 36.37	10.63	12.3	156.89	34.18	2000.658	

Table Notes:

1. Parallax primary: 8.29 ± 1.06 (Hipparcos). Spectral Type primary: F7V (Hipparcos), spectral type secondary: G (Tycho- 2 Spectral Type Catalog).
2. Parallax primary: 3.57 ± 0.91 (Hipparcos). Spectral Type primary: G8/K0IV (Hipparcos) .
3. Spectral Type primary: G5/8 (Tycho- 2 Spectral Type Catalog).
4. Parallax primary: 12.1 ± 1 (Hipparcos). Spectral Type primary: G5/G6 (Hipparcos) .
5. Parallax primary: 10.1 ± 0.72 (Hipparcos). Spectral Type primary: G8/K0 (Hipparcos) .
6. Parallax primary: 7.84 ± 0.71 (Hipparcos). Spectral Type primary: F0 V (Tycho- 2 Spectral Type Catalog).
7. Parallax primary: 7.75 ± 1.4 (Hipparcos). Spectral Type primary: K0 (Hipparcos) .
8. Parallax primary: 8.94 ± 0.89 (Hipparcos). Spectral Type primary: F6/F7V (Hipparcos) .
9. Spectral Type primary: G6 IV (Tycho- 2 Spectral Type Catalog).
10. Spectral Type primary: K5 (Tycho- 2 Spectral Type Catalog).
11. Spectral Type primary: F3/5 V (Tycho- 2 Spectral Type Catalog).
12. Parallax primary: 9.94 ± 0.94 (Hipparcos). Spectral Type primary: F2 (Hipparcos).
13. The secondary is TDS8273.
14. Parallax primary: 3.49 ± 1.25 (Hipparcos). Spectral Type primary: K1III (Hipparcos) .
15. Parallax primary: 13.2 ± 1.11 (Hipparcos).

(Continued on page 212)

New Common Proper-Motion Pairs from the PPMX Catalog

(Continued from page 211)

Spectral Type primary: F5 (Hipparcos), spectral type secondary: K2 (Tycho- 2 Spectral Type Catalog).

16. Spectral Type primary: F7/G0 (Tycho- 2 Spectral Type Catalog).
17. Spectral Type primary: F2/5 III/IV (Tycho- 2 Spectral Type Catalog).
18. Spectral Type primary: F8/G0 V (Tycho- 2 Spectral Type Catalog).
19. Spectral Type primary: F3/5 V (Tycho- 2 Spectral Type Catalog).
20. Spectral Type primary: F6/8 V (Tycho- 2 Spectral Type Catalog).
21. The secondary is TDS 819.
22. Parallax primary: 7.32 ±0.61 (Hipparcos). Spectral Type primary: F5 (Hipparcos), spectral type secondary: F8 (Tycho- 2 Spectral Type Catalog).
23. Spectral Type primary: G5 (Tycho- 2 Spectral Type Catalog).
24. Spectral Type primary: F8 (Tycho- 2 Spectral Type Catalog).
25. Parallax primary: 14.4± 1.16 (Hipparcos). Spectral Type primary: G3/G5V (Hipparcos).
26. Spectral Type primary: G0 (Tycho- 2 Spectral Type Catalog).
27. Spectral Type primary: G8 V (Tycho- 2 Spectral Type Catalog).
28. Spectral Type primary: G0 (Tycho- 2 Spectral Type Catalog).
29. Parallax primary: 15.2± 1.35 (Hipparcos). Spectral Type primary: G8V (Hipparcos).
30. Parallax primary: 14.95 ±0.93(Hipparcos), parallax secondary: 15.09 ± 1.21 (Hipparcos). Spectral Type primary: F3V (Hipparcos), secondary: G5V (Tycho- 2 Spectral Type Catalog)..
31. Spectral Type primary: G0 V (Tycho- 2 Spectral Type Catalog).
32. Parallax primary: 9.78± 1.1 (Hipparcos). Spectral Type primary: G0V (Hipparcos).
33. Spectral Type primary: G0 (Tycho- 2 Spectral Type Catalog).
34. Secondary is RST4724.
35. Parallax primary: 8.84 ± 1.09 (Hipparcos). Spectral Type primary: F2 (Hipparcos), spectral type secondary: F5 V (Tycho- 2 Spectral Type Catalog).

Table 3: Proper Motion of Each Component (mas/yr)

RA DEC	μ_1	μ_2	σ_1	σ_2
00 32 19.099 -21 50 33.69	(58.0, -2.6)	(63.8, 0.5)	(2.8, 3.1)	(2.8, 3.1)
00 55 33.213 -43 16 11.73	(14.3, -87.7)	(7.3, -89.7)	(2.2, 2.2)	(2.2, 2.2)
01 00 52.512 -18 56 57.08	(50.5, -7.8)	(53.2, -11.3)	(2.3, 2.3)	(2.3, 2.3)
01 52 27.571 -49 31 25.56	(31.2, -51.8)	(29.6, -50.5)	(1.5, 1.4)	(1.5, 1.4)
02 04 18.761 -70 59 40.88	(119.0, -29.9)	(124.9, -33.4)	(1.7, 2.0)	(1.9, 1.9)
02 05 27.334 +38 20 57.02	(33.1, -70.5)	(23.5, -83.2)	(2.4, 2.3)	(10.1, 10.6)
03 10 41.550 -20 06 41.54	(60.9, 3.4)	(65.7, 4.4)	(1.8, 1.8)	(2.4, 2.5)
03 24 54.681 -43 12 55.77	(3.0, 53.4)	(3.7, 58.6)	(1.8, 1.8)	(2.1, 2.1)
03 46 09.569 -41 12 22.33	(66.6, 25.0)	(71.9, 24.9)	(0.7, 0.8)	(2.1, 2.1)
04 23 48.192 -76 43 09.45	(36.4, -43.3)	(38.4, -43.5)	(1.4, 1.4)	(1.4, 1.4)
05 01 36.173 -44 49 49.05	(8.3, -65.8)	(5.2, -68.8)	(1.6, 1.6)	(2.1, 2.0)
05 23 39.978 -38 18 48.09	(29.5, 41.5)	(34.1, 37.2)	(2.3, 2.2)	(2.3, 2.3)

Table continues on next page.

New Common Proper-Motion Pairs from the PPMX Catalog

Table 3 (continued): Proper Motion of Each Component (mas/yr)

RA DEC	μ_1	μ_2	σ_1	σ_2
05 57 24.758 -40 23 51.78	(-1.3, 62.8)	(-3.0, 60.5)	(1.5, 1.5)	(2.3, 2.3)
06 15 01.664 -63 20 38.82	(-2.4, 84.8)	(-7.4, 83.7)	(3.8, 3.8)	(3.9, 3.9)
06 38 17.669 +18 28 24.58	(-78.3, -21.0)	(-73.4, -20.9)	(1.6, 1.6)	(1.6, 1.6)
07 03 08.446 -73 50 13.91	(-7.6, 64.2)	(-5.1, 57.0)	(6.0, 6.3)	(1.9, 1.9)
07 08 55.244 -11 23 29.58	(-5.5, -74.1)	(-2.4, -73.0)	(1.6, 1.7)	(2.0, 2.1)
09 33 19.911 -07 11 24.75	(-59.3, -26.5)	(-59.9, -23.8)	(1.5, 1.7)	(2.2, 2.2)
09 51 08.004 -18 39 31.43	(-97.4, 26.1)	(-101.1, 21.8)	(1.7, 1.6)	(3.0, 3.0)
10 14 38.104 -13 33 29.10	(-50.9, -17.8)	(-52.5, -21.1)	(1.3, 1.3)	(1.6, 1.6)
10 15 08.028 -65 26 11.04	(-62.0, 13.9)	(-60.0, 15.2)	(2.2, 2.2)	(2.2, 2.2)
10 16 15.352 -17 11 13.12	(-94.0, -27.8)	(-94.1, -35.4)	(2.6, 2.7)	(2.5, 2.5)
10 32 03.297 -30 28 05.49	(28.3, -42.4)	(38.2, -35.9)	(4.1, 4.1)	(3.4, 3.4)
10 47 27.741 -11 54 08.72	(21.8, -63.6)	(20.0, -64.3)	(1.5, 1.5)	(1.9, 2.0)
11 29 03.259 -38 17 05.77	(-9.3, -70.4)	(-12.1, -68.8)	(2.4, 2.5)	(2.9, 2.9)
11 35 52.047 -40 40 36.43	(-84.5, 8.0)	(-78.7, 10.5)	(2.8, 2.8)	(2.9, 2.9)
11 53 22.088 -67 07 05.56	(-9.1, -55.5)	(-5.9, -56.8)	(2.3, 2.3)	(2.4, 2.4)
12 05 25.156 +17 17 21.69	(-62.8, -53.3)	(-63.9, -51.8)	(1.1, 1.2)	(1.2, 1.3)
12 13 30.463 -48 47 46.64	(37.1, -54.7)	(33.7, -53.0)	(1.5, 1.5)	(2.0, 2.0)
12 34 19.535 -35 22 46.48	(-60.9, 8.7)	(-63.7, 3.6)	(2.5, 2.5)	(2.8, 2.8)
12 35 15.748 -09 10 57.84	(-49.4, -13.5)	(-48.5, -13.9)	(1.8, 1.9)	(1.9, 1.9)
12 35 43.067 -03 00 58.10	(-76.3, 13.9)	(-76.1, 14.3)	(0.7, 0.6)	(1.5, 1.5)
12 36 16.407 -79 31 34.45	(-58.2, -11.7)	(-60.5, -5.6)	(1.6, 1.6)	(2.0, 2.0)
13 17 35.429 -11 57 01.34	(-85.6, 35.8)	(-82.6, 36.2)	(1.7, 1.7)	(2.1, 2.1)
13 53 54.390 -07 45 44.87	(-48.1, -26.1)	(-45.8, -28.9)	(2.0, 2.0)	(2.7, 2.7)
14 08 01.294 -13 16 09.22	(-68.9, 0.9)	(-55.9, -15.5)	(9.7, 12.0)	(2.1, 2.1)
14 12 31.776 -30 06 17.68	(-72.7, -15.6)	(-67.0, -23.3)	(2.7, 2.7)	(2.7, 2.7)
14 35 32.025 -35 26 39.17	(-41.8, -28.4)	(-43.9, -25.8)	(2.3, 2.4)	(2.4, 2.4)
14 37 23.205 -66 50 27.83	(-56.1, -14.1)	(-55.0, -16.2)	(1.9, 1.9)	(1.8, 1.8)
14 53 52.152 -63 53 53.17	(86.2, -26.2)	(85.7, -25.7)	(2.7, 2.7)	(3.4, 3.2)
14 55 28.254 -56 48 55.15	(-46.1, -37.2)	(-51.6, -41.0)	(2.1, 2.1)	(2.2, 2.2)
14 58 31.045 -27 24 06.18	(13.8, -104.4)	(8.9, -104.8)	(3.7, 3.5)	(3.5, 3.4)
15 04 08.091 -26 23 26.83	(56.8, -8.6)	(52.8, -4.9)	(3.5, 3.2)	(2.7, 2.5)

Table continues on next page.

New Common Proper-Motion Pairs from the PPMX Catalog

Table 3 (continued): Proper Motion of Each Component (mas/yr)

RA DEC	μ_1	μ_2	σ_1	σ_2
15 07 12.834 -41 41 31.12	(-44.4, -25.3)	(-46.0, -28.6)	(1.8, 1.8)	(2.4, 2.4)
15 13 59.433 -58 37 15.68	(-61.8, -45.9)	(-62.2, -42.9)	(1.6, 1.6)	(2.4, 2.4)
16 31 42.851 +70 55 59.84	(-65.0, 22.5)	(-63.5, 23.4)	(1.3, 1.4)	(1.9, 1.9)
16 37 35.305 +69 19 17.25	(-82.0, 103.8)	(-82.4, 105.4)	(1.3, 1.4)	(1.8, 1.8)
17 01 49.674 +14 42 27.70	(-17.9, -47.2)	(-19.0, -47.1)	(1.6, 1.7)	(0.9, 0.9)
17 40 21.442 +05 43 37.44	(-24.4, -74.9)	(-25.2, -68.4)	(2.9, 2.9)	(2.9, 2.9)
17 54 51.203 +28 51 38.93	(-20.1, -47.0)	(-23.5, -45.9)	(1.7, 1.7)	(1.7, 1.7)
17 56 59.673 -46 06 31.92	(70.6, -20.9)	(72.6, -19.3)	(1.8, 1.8)	(1.8, 1.8)
17 59 53.975 -45 17 20.72	(5.9, -55.5)	(6.4, -53.5)	(2.1, 2.2)	(2.2, 2.3)
18 07 28.944 +00 29 27.19	(-18.2, -48.3)	(-18.4, -49.4)	(1.8, 1.8)	(1.8, 1.8)
18 13 05.162 +18 40 45.60	(5.4, -50.6)	(5.1, -52.0)	(1.1, 1.1)	(1.6, 1.6)
18 21 26.185 -15 22 18.08	(12.8, -82.3)	(12.2, -82.8)	(1.7, 1.8)	(1.8, 1.8)
18 22 44.038 -40 44 59.30	(-32.4, -38.8)	(-31.2, -40.2)	(2.2, 2.2)	(2.3, 2.4)
18 27 24.762 +21 51 53.42	(-16.4, 53.7)	(-18.0, 50.4)	(1.5, 1.5)	(2.2, 2.2)
18 38 36.531 +49 00 42.13	(69.8, 43.1)	(65.8, 42.7)	(1.4, 1.4)	(1.1, 1.1)
18 41 25.436 -44 32 30.63	(16.7, -51.9)	(14.9, -56.7)	(2.2, 2.2)	(2.2, 2.2)
18 45 04.604 -23 15 07.22	(17.7, -119.5)	(21.4, -126.1)	(1.7, 1.6)	(2.8, 2.8)
18 54 43.138 -50 07 46.85	(-2.6, -76.9)	(-1.7, -75.8)	(1.5, 1.5)	(2.2, 2.2)
19 01 33.281 -24 08 28.08	(-25.6, -83.0)	(-30.1, -82.5)	(1.6, 1.5)	(2.5, 2.6)
19 07 06.084 -14 04 09.97	(94.3, 27.6)	(91.4, 28.0)	(1.6, 1.6)	(2.2, 2.1)
20 06 03.948 -41 37 36.45	(59.1, -53.6)	(58.6, -57.6)	(2.1, 2.1)	(2.1, 2.1)
20 33 53.250 -27 10 17.31	(68.4, -82.3)	(71.2, -89.2)	(1.6, 1.6)	(3.3, 3.3)
20 36 05.730 -67 05 22.53	(-29.0, -63.8)	(-28.6, -67.9)	(1.7, 1.7)	(1.8, 1.8)
20 46 51.514 -49 28 39.62	(71.9, -2.3)	(74.1, -1.0)	(2.1, 2.2)	(2.2, 2.2)
21 10 18.359 -13 04 05.84	(73.1, -41.2)	(65.6, -42.8)	(5.2, 6.4)	(2.1, 2.1)
21 16 13.422 -40 40 51.94	(45.3, -27.1)	(46.0, -27.0)	(1.8, 1.8)	(1.8, 1.8)
21 29 36.229 -44 13 50.10	(86.8, -53.0)	(86.4, -49.2)	(2.0, 2.0)	(2.0, 2.1)
21 36 58.092 -35 53 03.02	(90.9, -10.1)	(90.7, -9.2)	(0.8, 0.5)	(1.4, 1.4)
21 54 22.594 -44 09 46.37	(13.7, -89.1)	(18.8, -92.5)	(2.1, 2.2)	(3.1, 3.1)
22 08 27.535 -57 06 52.51	(10.6, -73.0)	(11.0, -68.4)	(2.3, 2.3)	(2.4, 2.4)
22 09 42.632 -33 45 15.41	(49.3, -84.1)	(47.8, -82.7)	(1.5, 1.4)	(2.4, 2.4)

Table concludes on next page.

New Common Proper-Motion Pairs from the PPMX Catalog

Table 3 (continued): Proper Motion of Each Component (mas/yr)

RA DEC	μ_1	μ_2	σ_1	σ_2
22 32 09.402 -13 35 51.81	(-19.4, -65.9)	(-20.1, -62.8)	(1.6, 1.6)	(1.5, 1.5)
22 33 45.700 +61 45 26.85	(-18.6, -61.7)	(-15.8, -58.4)	(2.3, 2.1)	(2.0, 2.0)
22 41 49.606 +59 47 35.64	(16.9, -79.0)	(15.4, -77.5)	(1.2, 1.3)	(1.6, 1.6)
22 47 55.497 +03 36 07.26	(67.9, 3.3)	(69.6, 3.3)	(1.9, 1.9)	(1.6, 1.6)
22 53 55.679 -37 09 40.50	(108.1, 5.4)	(106.4, -0.8)	(1.4, 1.6)	(3.1, 3.1)
22 54 19.523 +30 22 18.31	(142.7, -4.6)	(142.0, -9.8)	(1.5, 1.5)	(1.6, 1.6)
23 28 08.467 -02 26 53.36	(69.8, 10.4)	(70.5, 8.4)	(1.2, 1.3)	(1.3, 1.4)
23 37 40.086 +00 46 36.37	(-30.0, -69.7)	(-29.8, -85.9)	(1.7, 1.8)	(10.7, 10.7)

(Continued from page 208)

advantages of this approach are:

- The basic core operations are easier to test and debug than the usual complex operations required by the data mining process.
- Assuming that the basic operations have been thoroughly checked, the whole process becomes less prone to errors. This holds true because the different filters are now written in a higher abstraction level, instead of directly in SQL. This makes them more understandable and easier to test and modify.
- The same user-scripts can be employed in different projects, facilitating the reusability of the code. The language allows defining parameterized functions with this purpose.
- This possibility of easily modifying the code is very useful for proving variations of the same filter and designing new ones. In our case it has been crucial for comparing the two versions of the first Halbwachs' condition examined above.

Currently, we are improving the application in order to make it publicly available in the near future. Regarding our test-case, the PPMX catalog, we would like to point out two main conclusions:

- The catalog was chosen to check the software, assuming that all the interesting pairs had been already extracted. Indeed most of them were in the WDS, but still there were a few possible interesting pairs to find.
- In all the projects examined up to now many of the pairs detected by data mining did not exist in the plates. Instead they corresponded to erroneous data obtained while processing the images. However, in this catalog no false pairs were found, attesting to the quality of its data.

As usual, it is important to remark that we don't claim that the CPMPs found are true binaries. The data mining process only suggests that these pairs might deserve more measurements and a deeper study.

Acknowledgements

This research makes use of the ALADIN Interactive Sky Atlas and of the VizieR database of astronomical catalogs, all maintained at the Centre de Données Astronomiques, Strasbourg, France, and of the data products from the Two Micron All Sky Survey, which is a joint project of the University of Massachusetts and the Infrared Processing and Analysis Center/California Institute of Technology, funded by the National Aeronautics and Space Administration and the National Science Foundation. This work has been partially supported by the Spanish projects TIN2008-06622-C03-01, S-0505/TIC/0407, S2009TIC-1465 and UCM-BSCH-GR58/08-910502.

References

Allende Prieto, C; Dambert D.L.; 1999, "Fundamental parameters of nearby stars from the comparison with evolutionary calculations: masses, radii and effective temperatures". *Astronomy and Astrophysics*, 352, p.555-562 (1999)

New Common Proper-Motion Pairs from the PPMX Catalog

Bonnarel, F.; Fernique, P.; Bienaymé, O.; Egret, D.; Genova, F.; Louys, M.; Ochsenbein, F.; Wenger, M.; Bartlett, J. G.; 2000, "The ALADIN interactive sky atlas. A reference tool for identification of astronomical sources", *Astronomy and Astrophysics Supplement*, 143, p.33-40.

Caballero, R.; 2009, "Finding New Common Proper-Motion Binaries by Data Mining", *Journal of Double Star Observations*, 5(3), 156-167.

Caballero, R.; 2010, "New Northern Hemisphere Common Proper-Motion Pairs from the UCAC-3 Catalog", *Journal of Double Star Observations*, 6(1), 97-113.

Greaves, J., 2004, "New Northern hemisphere common proper-motion pairs". *Monthly Notices of the Royal Astronomical Society* 355, 585-590.

Halbwachs, J.L., 1986, "Common proper motion stars in the AGK3". *Bull. Inf. Centre Donnees Stellaires*, 30, p.129.

Mason, B. D.; Wycoff, G.; Hartkopf, W. I., 2003, "The Washington Double Star Catalog", http://ad.usno.navy.mil/proj/WDS/wds.html

Perryman, M. A. C.; ESA, 1997, *The HIPPARCOS and TYCHO catalogues. Astrometric and photometric star catalogues derived from the ESA HIPPARCOS Space Astrometry Mission*, Publisher: Noordwijk, Netherlands: ESA Publications Division, 1997, Series: ESA SP Series vol no: 1200, ISBN: 9290923997.

Röser, S.; Schilbach, E.; Schwan, H.; Kharchenko, N. V.; Piskunov, A. E.; Scholz, R.-D.; 2008 "PPM-Extended (PPMX) – a catalogue of positions and proper motions". *Astronomy and Astrophysics*, 488 (1), 401-408.

Salim, S.; Gould, A., 2003, "Improved Astrometry and Photometry for the Luyten Catalog. II. Faint Stars and the Revised Catalog", *The Astrophysical Journal*, 582, 1011-1031.

Wright, Candace O.; Egan, Michael P.; Kraemer, Kathleen E.; Price, Stephan D.; 2003, "The Tycho-2 Spectral Type Catalog", *The Astronomical Journal*, 125(1), 359-363.

Una aplicación para descubrir nuevos pares de estrellas con movimiento propio común mediante técnicas de minería de datos

- Blanca Collado Iglesias
- Antonio Javier Fernández Sánchez
- Sara Pozuelo González

CPMP es una aplicación cuyo objetivo es automatizar el proceso de minería de datos que realizamos para buscar nuevos pares de estrellas con movimiento propio común (Common Proper Motion Pairs)

CPMP is a special-purpose software automatizing and simplifying the detection of new common proper motion pairs of stars by using data mining techniques.

Introducción

Las técnicas de minería de datos han probado su utilidad para el descubrimiento de nuevos pares de movimiento propio común. La idea es partir de catálogos de estrellas disponibles de forma libre on-line que contengan información acerca del movimiento propio, y tratar de relacionar pares con movimiento similar y cumpliendo ciertos criterios físicos y estadísticos.

Sin embargo esta labor resulta repetitiva y tediosa: hay que importar los catálogos, inicialmente en formato texto, desde una base de datos, programar los criterios, etc. Además algunas de las tareas como la de cruzar una tabla consigo misma para encontrar pares cercanos, puede ser muy lenta si se programa en lenguajes de consulta como SQL. Todo ello hace interesante disponer de un software específico dedicado a estas tareas y que permita repetir el proceso de forma semi-automática.

La aplicación que damos a conocer en este artículo ha sido desarrollada por los autores de este artículo dentro del marco de la asignatura Sistemas Informáticos, impartida en la Facultad de Informática de la Universidad Complutense de Madrid y tutorizada por el profesor Rafael Caballero Roldán.

Descripción de la aplicación

Al iniciar *CPMP* se nos preguntara nuestro Nombre de Usuario y contraseña para conectarnos a la base de datos. En esta versión inicial, CPMP trabaja exclusivamente con el servidor de bases de datos MySQL que deberá estar instalado previamente.

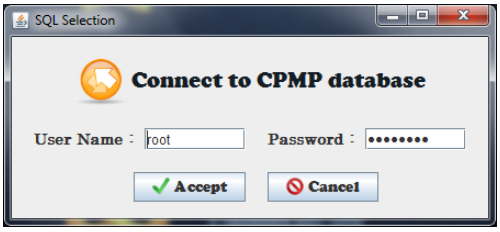


Ilustración 1

Una vez conectados correctamente se nos mostrará la pantalla desde la que podremos manejar todas las opciones que CPMP ofrece. (**Ilustración 2**)

Ilustración 2

En ella podemos observar las operaciones principales de las que consta nuestra herramienta. La primera opción **Import New Catalogue** es la operación básica antes de poder empezar a realizar la labor de minería de datos, ya que es la que nos permitirá cargar los datos de las estrellas individuales a la aplicación. Dichos datos deberán ser catálogos previamente descargados de Vizier (http://vizier.u-strasbg.fr/viz-bin/VizieR) en formato de texto y con los campos separados por ";".

El proceso de importar catálogos puede ser tedioso para el usuario, puesto que el tiempo necesario para realizar dicho proceso es lineal con respecto al tamaño del fichero de entrada (que puede llegar a ocupar varios gigas de memoria). Por ello, la aplicación le permitirá añadir varios catálogos a la vez (**Ilustración 3**), realizando esta labor en un segundo plano y permitiéndonos así seguir trabajando con catálogos previamente añadidos.

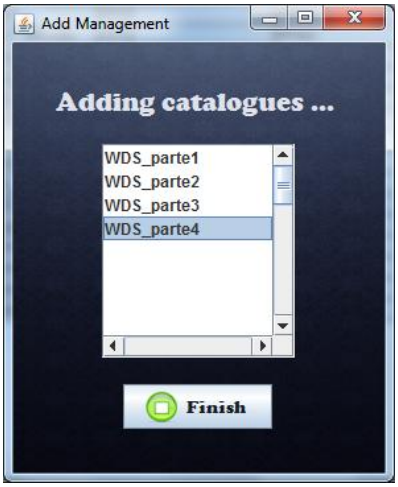


Ilustración 3

Según vaya finalizando el proceso de importar cada catálogo nos será mostrado un informe con la siguiente información: número de estrellas añadidas, número de estrellas no añadidas por estar duplicadas en el fichero de entrada y el tiempo requerido en realizar el proceso. (**Ilustración 4**)

Ilustración 4

Una vez que hemos importado los catálogos con los que queremos realizar la labor de minera de datos, ya tiene sentido que empecemos a explorar las demás funcionalidades de nuestra herramienta.

Si pulsamos sobre el botón **General Settings** cambiará la vista de la aplicación al panel mostrado en la **Ilustración 5**.

En él veremos listados los catálogos que hemos añadido anteriormente y las operaciones más generales que podemos realizar sobre ellos (**Tabla 1**).

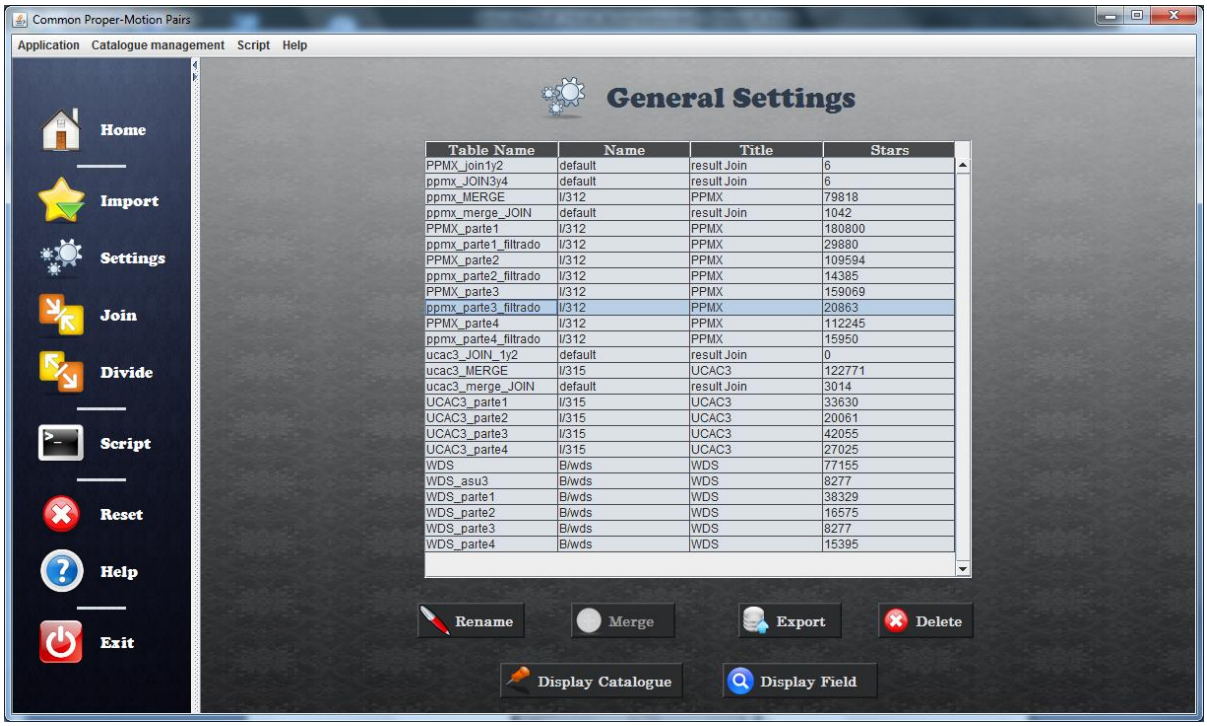


Ilustración 5

Nombre Operación	Descripción
Rename	Renombrar un catálogo, conservando el catálogo original y obteniendo una copia del mismo con el nuevo nombre.
Export	Podremos exportar un catálogo almacenado en nuestra base de datos a un fichero de texto.
Delete	Eliminar un catálogo.
Merge	Nos permitirá unir dos o más catálogos siempre que tenga los mismos campos (por ejemplo las distintas partes de un catálogo).
Display Field	Nos mostrara la lista de campos del catálogo seleccionado. *(ver **Ilustración 6**)*
Display Catalogue	Nos mostrara el contenido del catálogo *(ver **Ilustración 7**)*

Tabla 1

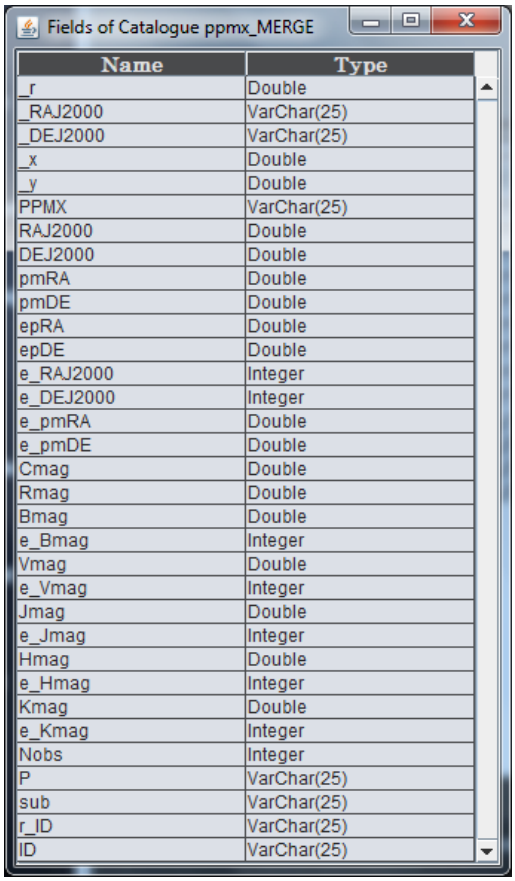


Ilustración 6

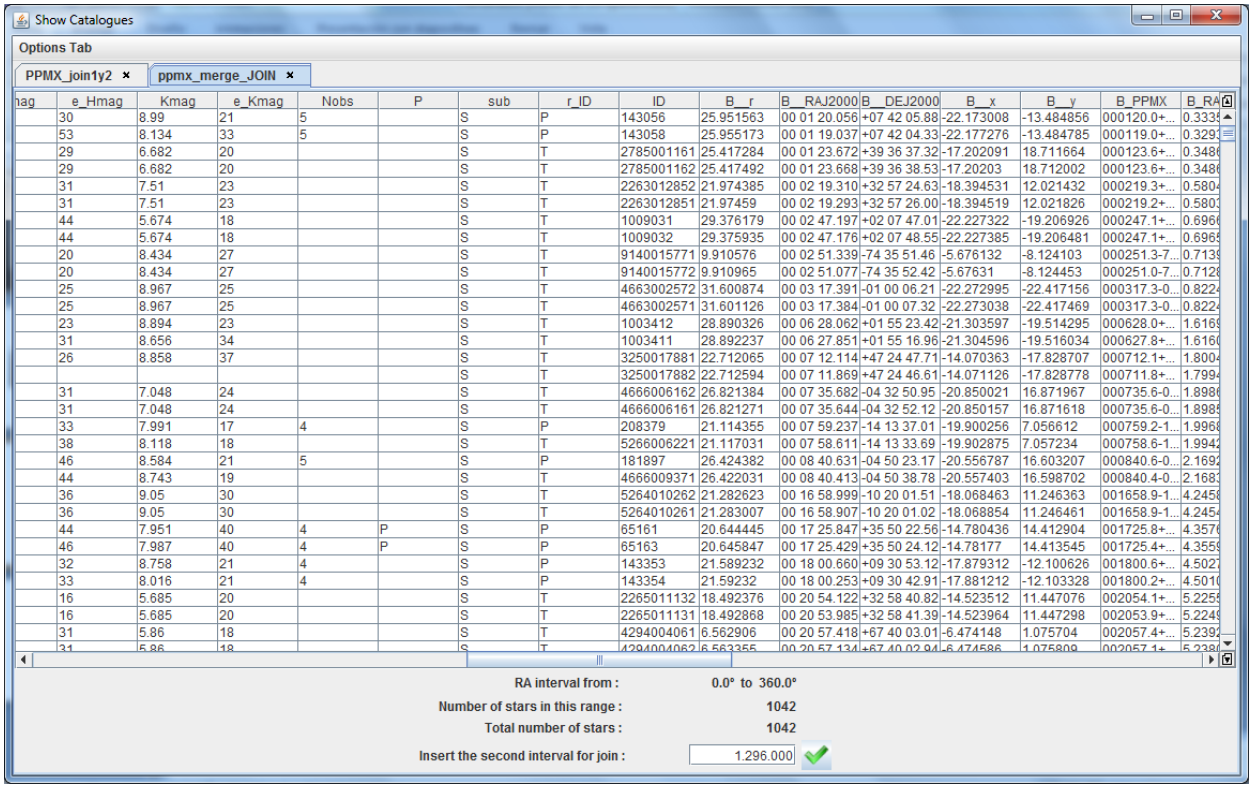


Ilustración 7

Para empezar con la propia labor de minería de datos, una de las opciones más útiles que nos encontramos en CPMP es la de **Join Catalogues** . Con esta opción el usuario podrá cruzar dos catálogos, creando una nueva tabla (catálogo de pares) que contiene por cada fila un par de estrellas cuya separación es menor que un parámetro en segundos *(ver **Ilustración 8**)*. Una de las aplicaciones más comunes de esta opción es cruzar una tabla consigo misma para obtener una lista de pares candidatos.

Sin embargo, la característica más importante y potente de esta herramienta es la que permite al usuario definir sus propios scripts para filtrar los pares de candidatas. Dichos scripts pueden ser guardados en formato de texto y son definidos mediante una serie de operaciones básicas. *(ver **Tabla 2**)* . Los scripts nos permitirán aplicar las mismas operaciones a distintos proyectos. La ventaja de esta funcionalidad es doble:

- Por un lado evitamos errores de programación ya que depuramos los scripts una sola vez.
- Al estar escritos en un lenguaje de alto nivel, podemos probar fácilmente variaciones que nos ayudarán a refinar los criterios utilizados para la detección de nuevos pares.

Además las operaciones básicas ya están incluidas en las operaciones accesibles al usuario mediante click de ratón (merge y rename en General Settings y Join desde su propia interfaz Join Catalogues).

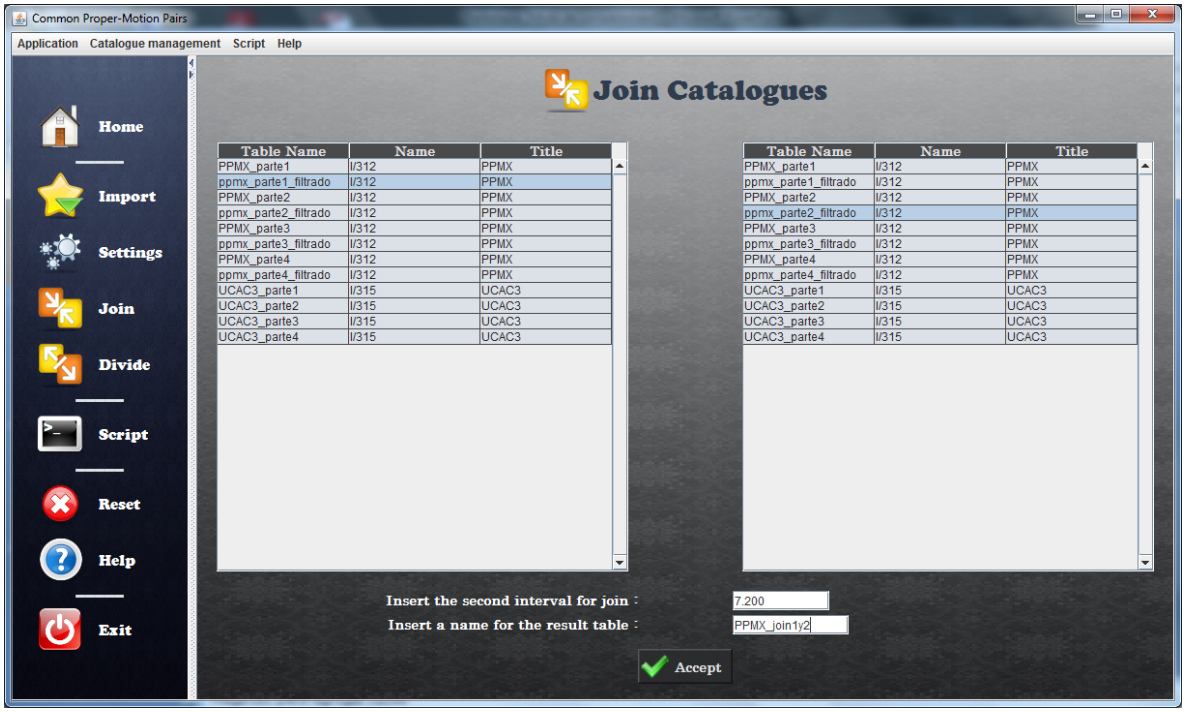


Ilustración 8

Descripción	Sintaxis
Combinar dos tablas	TablaNueva ← **merge**(tabla1,tabla2)
Renombrar una tabla	TablaNueva ← **rename**(tabla)
Cruzar dos tablas obteniendo una tabla de pares	TablaNueva ← **join**(tabla1,tabla2)[segundos]
Filtrar una tabla eliminando las filas que no cumplan la condición dada	TablaNueva ← **filter**(tabla)[condicionSQL]
Eliminar de una tabla de pares las parejas que están a mas de una distancia dada	TablaNueva ← **distance**(tabla)[segundos]
Añadir un campo nuevo a una tabla	TablaNueva ←**newAttribute**(tabla,AtributeName,AttributeType)
Dar valor a un campo en todas las filas que	TablaNueva ← **attribute**(AttributeName,Value)[condición SQL]

cumplan una condición dada	
Eliminar un campo	TablaNueva ← **deleteAttribute**(tabla ,AttributeName)
Obtener una nueva tabla con solo aquellas filas de la tabla1 que no estén en la tabla2	TablaNueva ← **Minus**(Tabla1,Tabla2)

Tabla 2

Además el sistema permite definir funciones parametrizadas que serán muy útiles para definir las operaciones más comunes, de forma que puedan ser usadas en los diferentes scripts que nos queramos editar, o crear otras nuevas mucho más complejas a partir del repertorio básico ofrecido al usuario.

Para poder escribir nuestros scripts y funciones usaremos la opción **Script**.

La ventana que se muestra lo hace por defecto en el modo para poder editar y ejecutar un script. Si lo que queremos es crear una función parametrizada y guardarla deberemos cambiar a modo función (Options **-> Modo Function**).

En ambos modos tenemos visible la lista de catálogos disponibles, donde podremos observar los catálogos ya importados, así como los nuevos catálogos que se irán creando al ejecutar nuestros scripts.

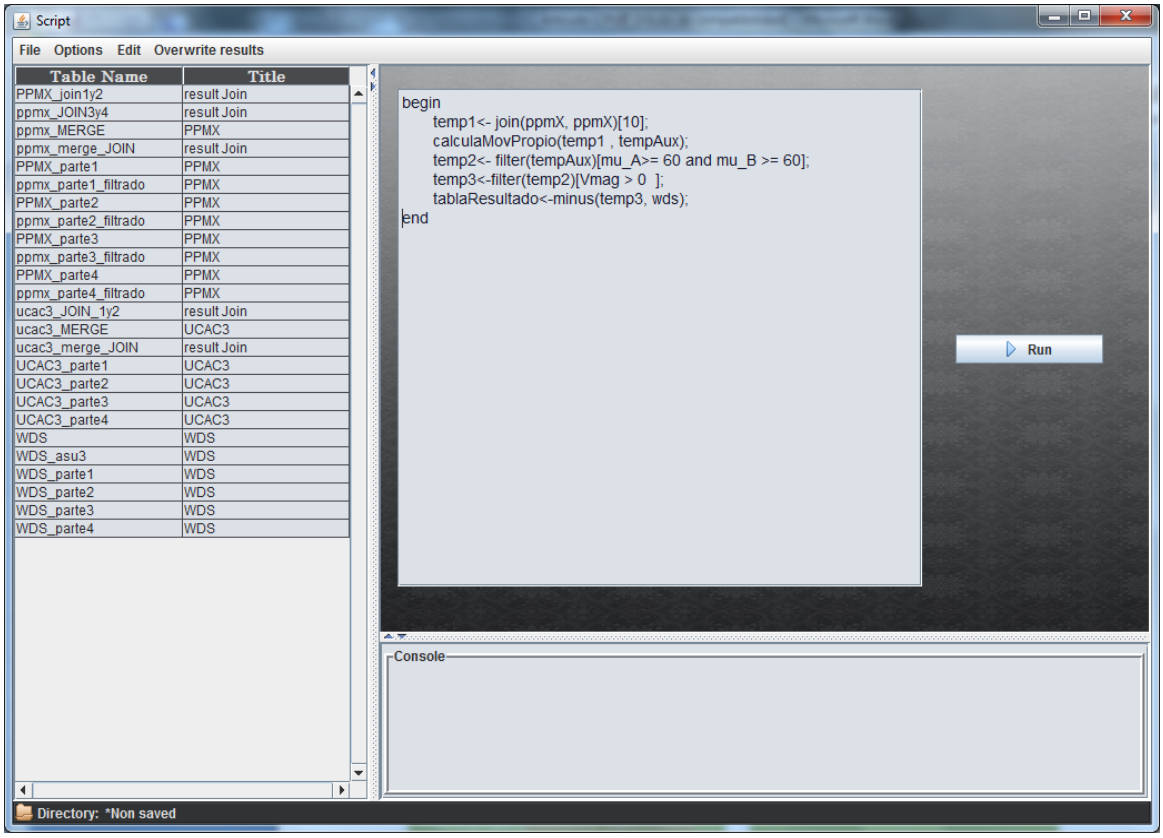


Ilustración 9

Por ejemplo un posible script en lenguaje natural para buscar pares de estrellas en el catálogo PPMX seria:

1. Buscar estrellas que se encuentre a menos de 10'' de separación
2. De las anteriores quedarnos con las que tengan un movimiento propio por encima de 60 ms / año
3. De las anteriores quedarnos solo con aquellas que tengan una magnitud V no vacía
4. Asegurarnos que las encontradas no pertenecen ya al catálogo WDS.

Esto escrito en nuestro lenguaje es:

```
begin
        temp1<- join(ppmX, ppmX)[10];
        calculaMovPropio(temp1 , tempAux);
        temp2<- filter(tempAux)[mu_A>= 60 and mu_B >= 60];
        temp3<-filter(temp2)[Vmag is not null  ];
        tablaResultado<-minus(temp3, wds);
end
```

Donde ***calculaMovPropio(temp1,tempAux)*** es una función que previamente hemos definido en el Modo Function :

```
Function calculaMovPropio(2)
begin
    aux1<- newAttribute($1,mu_A,double);
    aux2<- newAttribute(aux1,mu_B,double);
    aux3<- attribute(aux2, mu_A, ' sqrt(pmra*pmra + pmde*pmde) ') ;
    $2<- attribute(aux3, mu_B, ' sqrt(b_pmra*b_pmra + b_pmde* b_pmde) ') ;
end
```

Una vez escrito el script pulsamos el botón correspondiente a la opción **Run** y se ejecutará dicho programa. En la consola se irán mostrando los resultados de las distintas instrucciones intermedias y también nos informará si alguna operación no se pudo llevar a cabo, ya sea por un error de sintaxis o porque las tablas introducidas como parámetros no tienen el formato requerido o no existen.

Finalmente, si queremos comprobar cada par en las placas fotográficas disponibles en la aplicación Aladin (http://aladin.u-strasbg.fr/aladin.gml), podremos hacerlo fácilmente desde la propia aplicación. Para ello deberemos volver a la ventana General Settings y seleccionar la tabla obtenida (tablaResultado) pulsando **Display Catalogue**. Esta acción, como ya sabemos, nos mostrará una nueva ventana con el contenido de nuestra tablaResultado. Ahora simplemente debemos hacer doble click en la fila de la tabla que contiene el par de estrellas que queremos visualizar, y nos abrirá directamente en nuestro navegador web el applet de Aladin con las imágenes coordenadas astronómicas del par seleccionado.

Conclusiones

La aplicación ha demostrado su utilidad para la obtención de nuevos pares procedentes de los catálogos IPHAS-POSSI, PPMX y UCAC3. La **Ilustración 10** muestra una de las

dobles nuevas, localizada en coordenadas 03 55 11.44+54 38 32.7, obtenida a partir del catálogo IPHAS-POSSI la composición en rojo y azul de dos imágenes tomadas por el Observatorio del Monte Palomar con una diferencia de 40 años (1954, 1994) combinadas en dos colores, rojo y azul lo que permite apreciar el movimiento

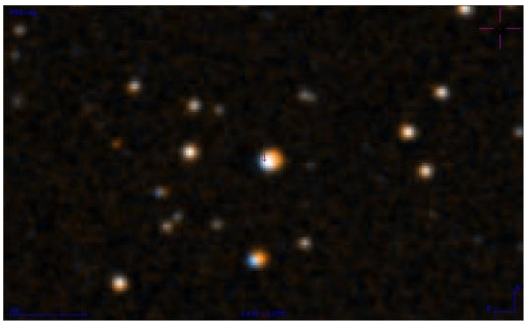


Ilustración 10

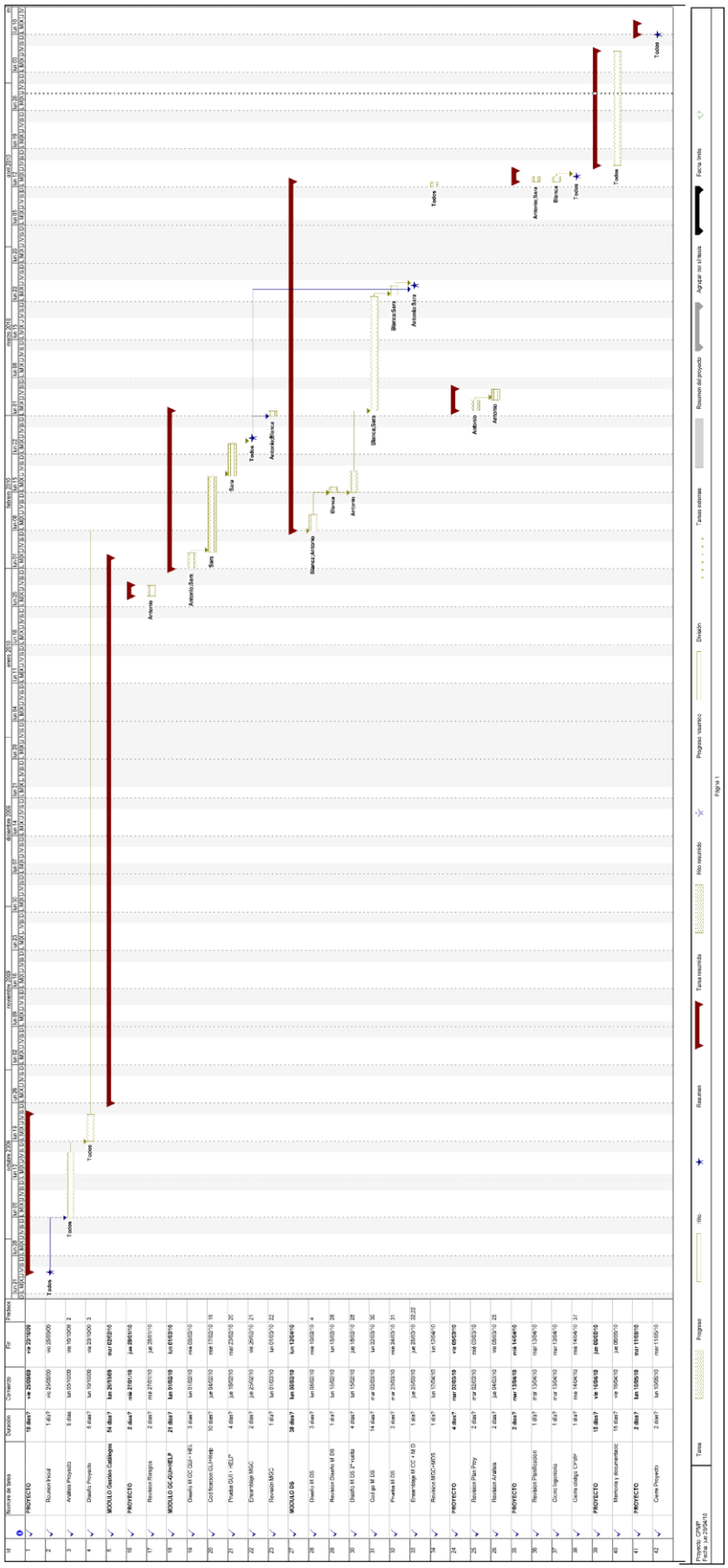
Al ser configurable mediante scripts creemos que la aplicación podría utilizarse con otros fines, como la detección de duplicados en el catálogo WDS.

APÉNDICE 4: ESTRUCTURA DE DESCOMPOSICIÓN DEL TRABAJO

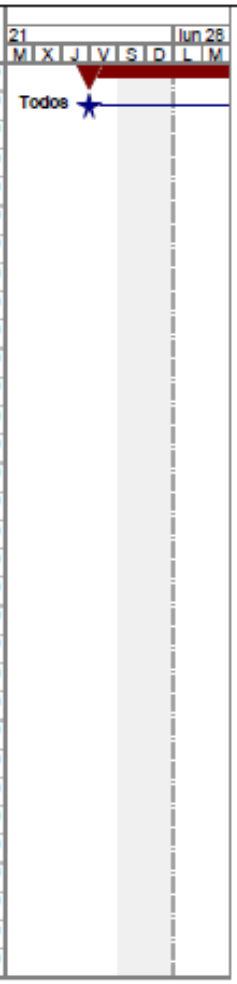
AE->	Com.Cliente	Plan. Gestión	Ingeniería		Const. Y Adapt			Evaluación
Acción			Análisis	Diseño	Codificion	Prueba	Ensamblaje	
PROYECTO	i:25/09/09 f:25/09/09 r: Todos e: Reunión inicial director proyecto		i: 5/10/2009 f: 16/10/2009 r: Todos e: Plan de Proyecto Software	i:19/10/2009 f:23/10/2009 r: Todos e: Especificación de requisitos software				
MÓDULO **Gestión catálogos**				i:26/10/2009 f:30/10/2009 r: Antonio e: Diseño mód.Gestión catálogos				i:02/11/2009 f:02/11/2009 r: Blanca, Sara e: Rev. Diseño móc. Gestión catálogos
MÓDULO **Gestión catálogos-núcleo funcional**				i:03/11/2009 f:05/11/2000 r: Antonio, Sara e: Diseño núcleo funcional	i: 10/11/2009 f: 10/12/2009 r: Antonio, Blanca e: Código núcleo funciona	i:14/12/2009 f:17/12/2009 r: Blaca, Sara e: prueba código nucleo funcional		I: 21/12/2009 f: 21/12/2009 r: Todos Rev. modulo núcleo funcional
MÓDULO Gestion catalogos-mineria datos				i:11/01/2010 f:12/01/2010 r: Antonio, Blaca. e: Diseño mineria de datos	i: 13/01/2010 f: 26/01/2010 r: Antonio, Sara e: Código mineria datos	i:27/01/2010 f:28/01/2010 r: Blanca, Sara e: prueba código mineria datos		I:1/02/2010 f:2/02/2010 r: Blanca Rev. mod. Mineria datos
PROYECTO	i:27/01/2010 f:28/01/2010 r: Antonio e: rev. plan y riesgos							
MÓDULO **Gestión Catálogos-interfaz gráfica más ayuda**				i:01/02/2010 f:03/02/2010 r: Antonio, Sara e: Diseño GUI+Help	i: 4/02/2010 f: 17/02/2010 r: Sara e: Código GUI+Help	i:18/02/2010 f:23/02/2010 r: Sara e: Prueba GUI+ Help	i:25/02/2010 f:26/02/2010 r: Todos e: ENS MOD. GC	i:01/03/2010 f:01/03/2010 r: Antonio, Blanca Rev. Módulo GC
PROYECTO	i:02/03/2010 f:03/03/2010 r: Antonio e:rev.plan proy	I:04/03/2010 f:05/03/2010 r: Antonio e:rev. anális.						
Módulo Desarrollo Scripts				i: 08/02/2010 f: 10/02/2010 r: Blanca, Antonio e: Diseño Ds				i:15/02/2010 f:15/02/2010 e:Blanca Rev. Diseño DS.
Módulo Desarrollo Scripts				i:15/02/2010 f:18/02/2010 r: Antonio e: Diseño Ds	i:02/03/2010 f:22/03/2010 r: Blanca, Sara e: Código Ds	i:23/03/2010 f:24/03/2010 r: Blanca, Sara. e: prueba código Ds	i:25/03/2010 f:25/03/2010 r: Antonio, Sara ENS. Gc+Ds	i:12/04/2010 f:12/04/2010 r:Todos Rev. Gc+ds
PROYECTO	i:13/04/2010 f:13/04/2010 r: Antonio, Sara e: Rev Planificación	i:13/04/2010 f:13/04/2010 r: Blanca e: Cierre Ing.	i:14/04/2010 f:14/04/2010 r:Todos e: Cierre Código CPMP					
PROYECTO	i:16/04/2010 f:06/05/2010 r: Todos e: Memoria							
PROYECTO	i:10/05/2010 f:11/05/2010 r: Todos e: Cierre Proyecto							

AE->	Com. Cliente	Plan. Gestión	Ingeniería		Const. Y Adapt			Evaluación	
Acción			**Análisis**	**Diseño**	**Codificación**	**Prueba**	**Ensamblaje**		**Total**
PROYECTO	Reunion Inicial **3pd**		Análisis proyecto **21pd**	Diseño proyecto **15 pd**					**39 pd**
MOD Gestión Catálogos				Diseño mod. Gestión catálogos **5 pd**				Rev- Diseño GC **2 pd**	**7 pd**
MOD GC-núcleo funcional				Diseño Núcleo fiuncional **6 pd**	Código Núcleo fiuncional **40 pd**	Prueba Núcleo fiuncional **9 pd**		Rev.Diseño núcleo funcional **3 pd**	**58 pd**
MOD GC-minería de datos				Diseño mod minería datos **4 pd**	Código mod minería datos **20 pd**	Prueba cod. minería datos **6 pd**		Rev.Diseño minería datos **2 pd**	**34 pd**
PROYECTO		Rev. Plan y riesgos **3 pd**							**3 pd**
MOD Interfaz gráfica + ayuda				Diseño mod interfaz gráfica **6 pd**	Código interfaz gráfica + ayuda **14 pd**	Prueba interfaz gráfica + ayuda **4 pd**	ENS MOD GC **6 pd**	Rev. Diseño GC **2 pd**	**32 pd**
PROYECTO		Rev. Plan proy **2 pd**	Rev. Plan análisis **2 pd**						**4 pd**
MOD Desarrollo Scripts				Diseño Ds **6 pd**				Rev. Diseño Ds **1 pd**	**7 pd**
MOD Desarrollo Scripts				Diseño Ds **4 pd**	Código DS **30 pd**	Prueba código MV **4 pd**	ENS. GC + DS **2 pd**	Rev. Diseño Ds **1 pd**	**41 pd**
PROYECTO		Rev. Planificación **2 pd**		Cierre ingeniería CPMP **1 pd**	Cierre código CPMP **3 pd**				**6 pd**
PROYECTO		Memoria **45 pd**							**45 pd**
PROYECTO		Cierre proy. CPMP **3 pd**							**3 pd**
									277 pd

APÉNDICE 6: DIAGRAMA DE GANTT



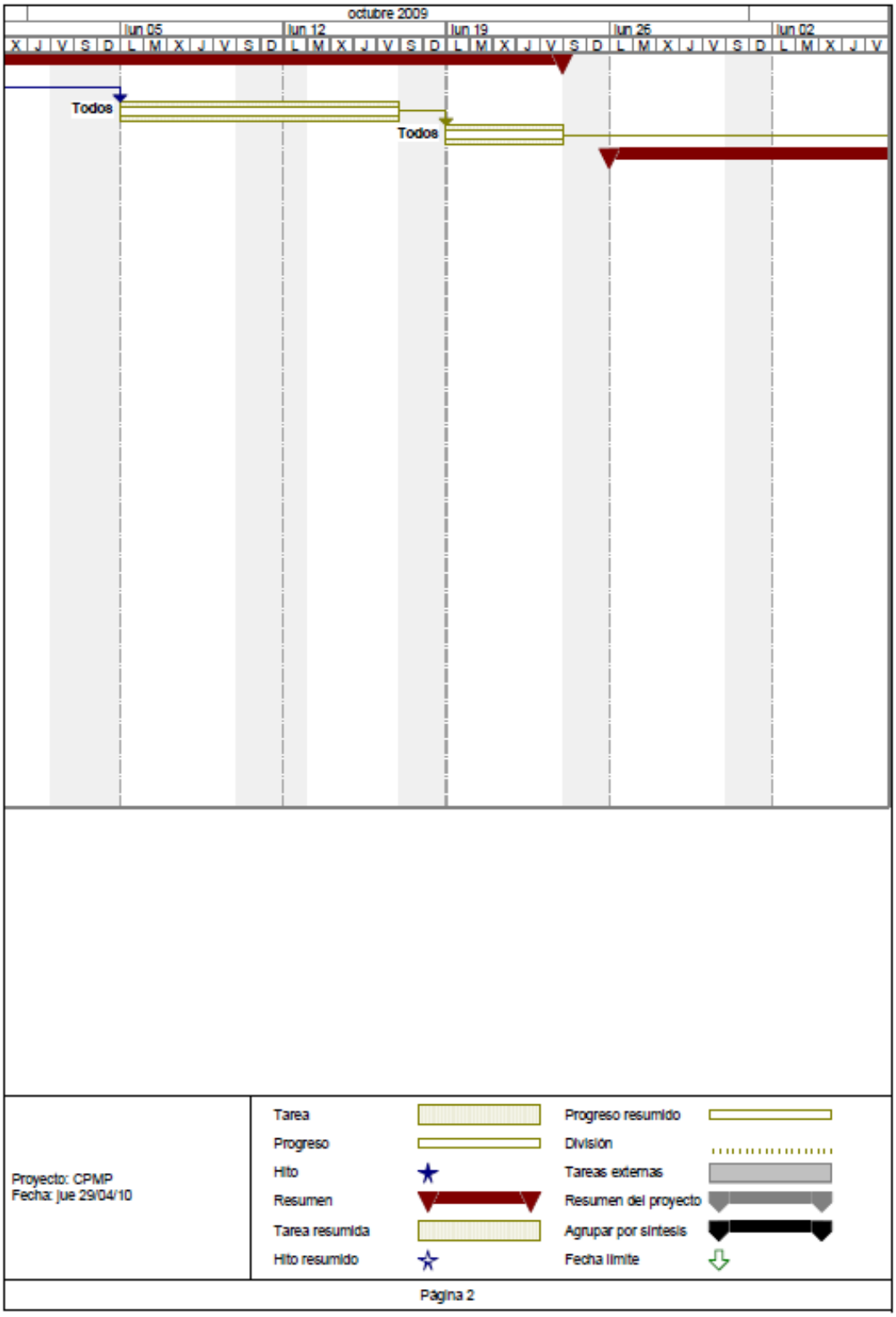
Id		Nombre de tarea	Duración	Comienzo	Fin
1	✓	**PROYECTO**	**19 días?**	**vie 25/09/09**	**vie 23/10/09**
2	✓	Reunion Inicial	1 día?	vie 25/09/09	vie 25/09/09
3	✓	Análisis Proyecto	9 días	lun 05/10/09	vie 16/10/09
4	✓	Diseño Proyecto	5 días?	lun 19/10/09	vie 23/10/09
5	✓	**MODULO Gestión Catálogos**	**54 días?**	**lun 26/10/09**	**mar 02/02/10**
16	✓	**PROYECTO**	**2 días?**	**mié 27/01/10**	**jue 28/01/10**
17	✓	Revision Riesgos	2 días?	mié 27/01/10	jue 28/01/10
18	✓	**MODULO GC-GUI+HELP**	**21 días?**	**lun 01/02/10**	**lun 01/03/10**
19	✓	Diseño M GC GUI+ HELP	3 días?	lun 01/02/10	mié 03/02/10
20	✓	Codificacion GUI+Help	10 días?	jue 04/02/10	mié 17/02/10
21	✓	Prueba GUI + HELP	4 días?	jue 18/02/10	mar 23/02/10
22	✓	Ensamblaje MGC	2 días?	jue 25/02/10	vie 26/02/10
23	✓	Revisión MGC	1 día?	lun 01/03/10	lun 01/03/10
27	✓	**MODULO DS**	**38 días?**	**lun 08/02/10**	**lun 12/04/10**
28	✓	Diseño M DS	3 días?	lun 08/02/10	mié 10/02/10
29	✓	Revision Diseño M DS	1 día?	lun 15/02/10	lun 15/02/10
30	✓	Diseño M DS 2ª vuelta	4 días?	lun 15/02/10	jue 18/02/10
31	✓	Codigo M DS	14 días?	mar 02/03/10	lun 22/03/10
32	✓	Prueba M DS	2 días?	mar 23/03/10	mié 24/03/10
33	✓	Ensamblaje M GC + M DS	1 día?	jue 25/03/10	jue 25/03/10
34	✓	Revision MGC+MDS	1 día?	lun 12/04/10	lun 12/04/10
24	✓	**PROYECTO**	**4 días?**	**mar 02/03/10**	**vie 05/03/10**
25	✓	Revision Plan Proy	2 días?	mar 02/03/10	mié 03/03/10
26	✓	Revision Analisis	2 días?	jue 04/03/10	vie 05/03/10
35	✓	**PROYECTO**	**2 días?**	**mar 13/04/10**	**mié 14/04/10**
36	✓	Revision Planificacion	1 día?	mar 13/04/10	mar 13/04/10
37	✓	Cierre Ingenieria	1 día?	mar 13/04/10	mar 13/04/10
38	✓	Cierre código CPMP	1 día?	mié 14/04/10	mié 14/04/10
39	✓	**PROYECTO**	**15 días?**	**vie 16/04/10**	**jue 06/05/10**
40	✓	Memoria y documentacion	15 días?	vie 16/04/10	jue 06/05/10
41	✓	**PROYECTO**	**2 días?**	**lun 10/05/10**	**mar 11/05/10**
42	✓	Cierre Proyecto	2 días?	lun 10/05/10	mar 11/05/10

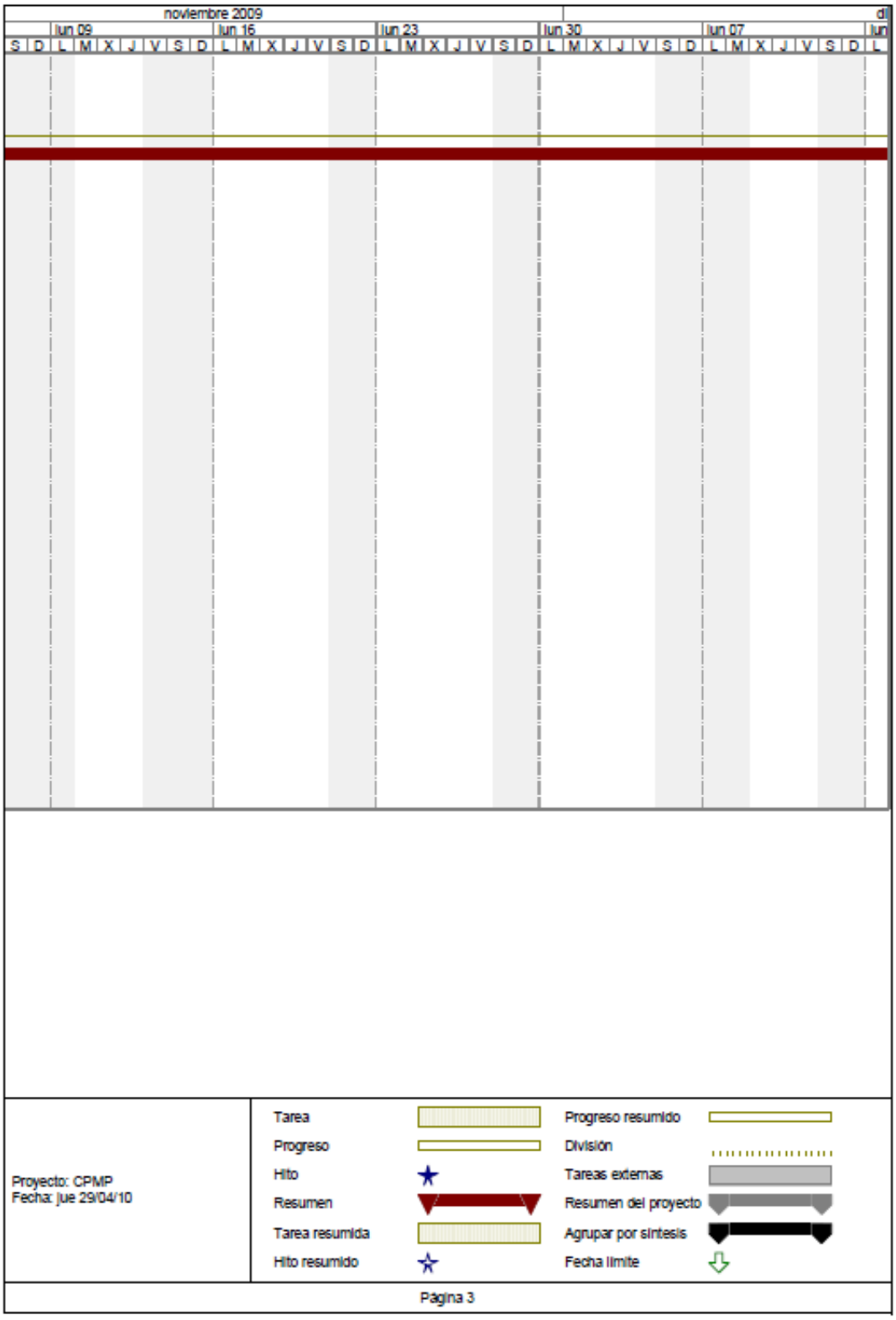


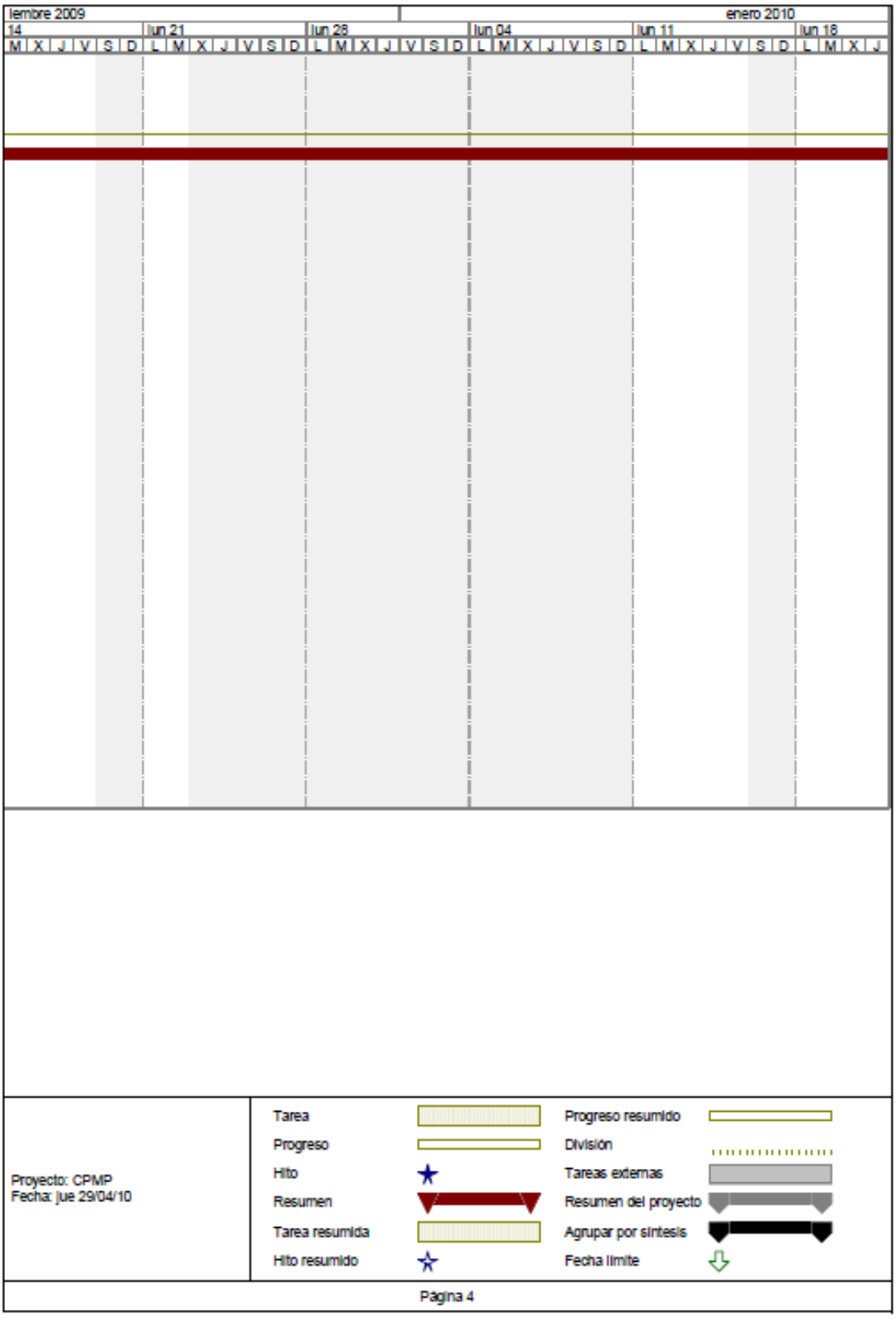
Proyecto: CPMP
Fecha: jue 29/04/10

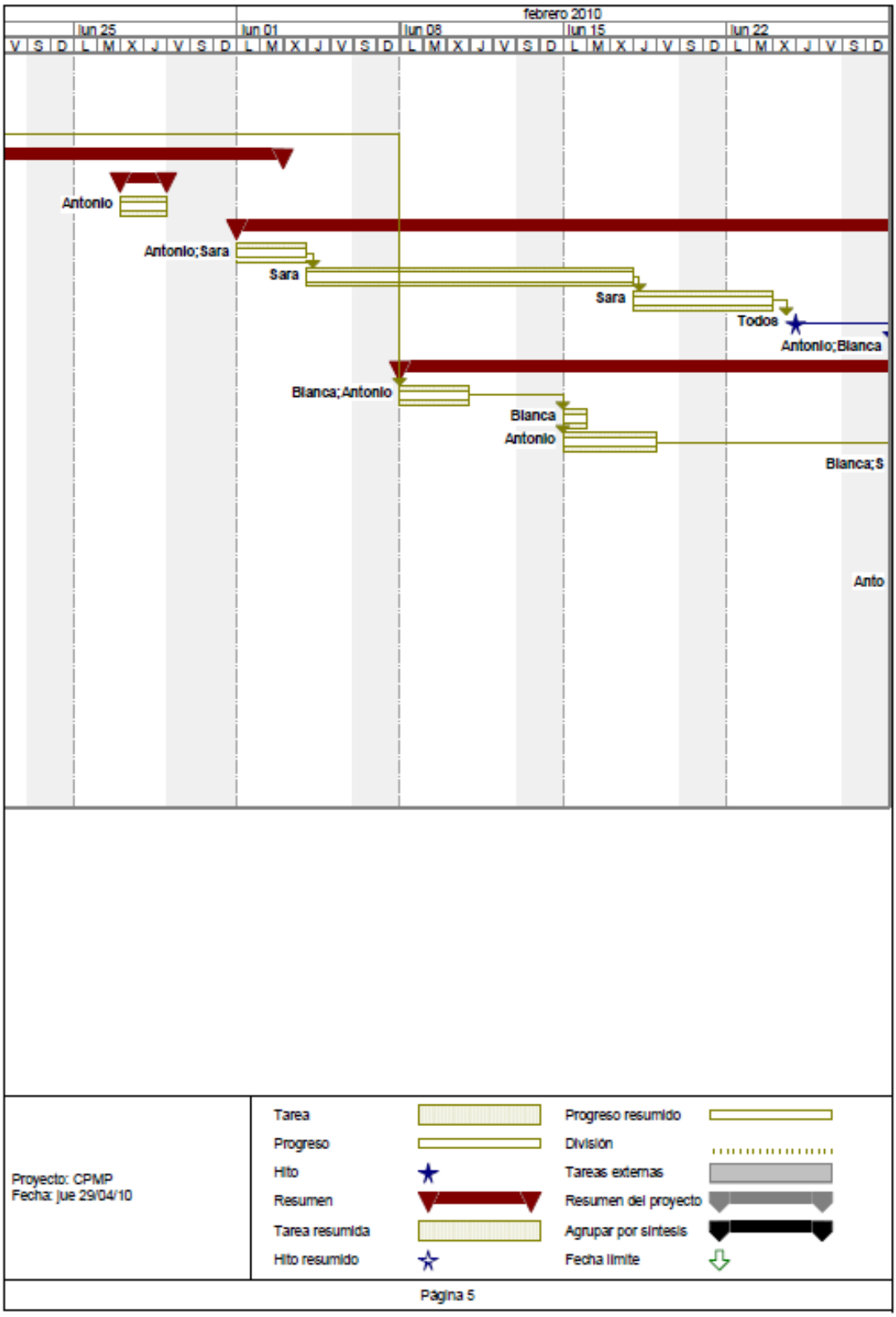
Tarea		Progreso resumido	
Progreso		División	
Hito		Tareas externas	
Resumen		Resumen del proyecto	
Tarea resumida		Agrupar por síntesis	
Hito resumido		Fecha límite	

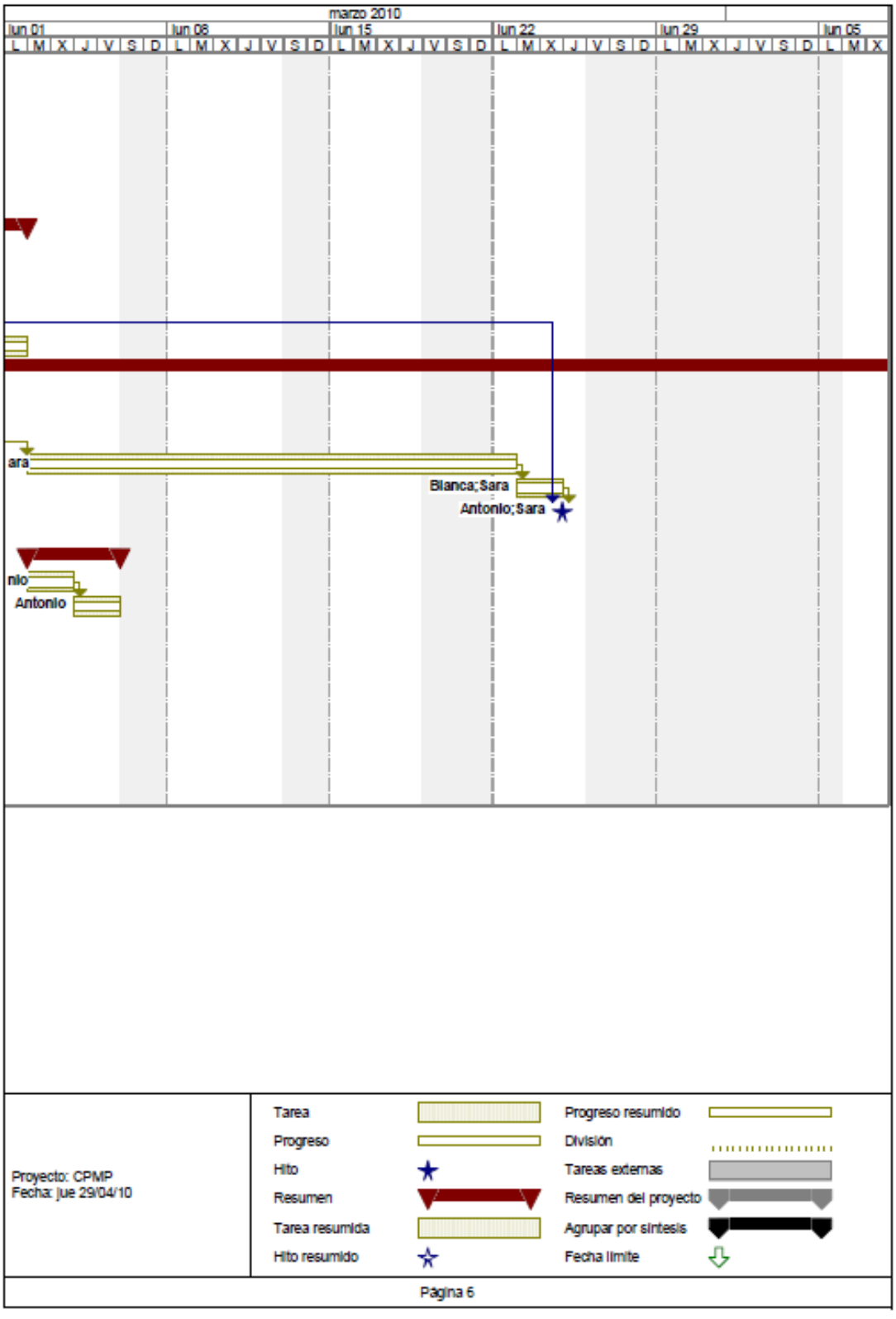
Página 1

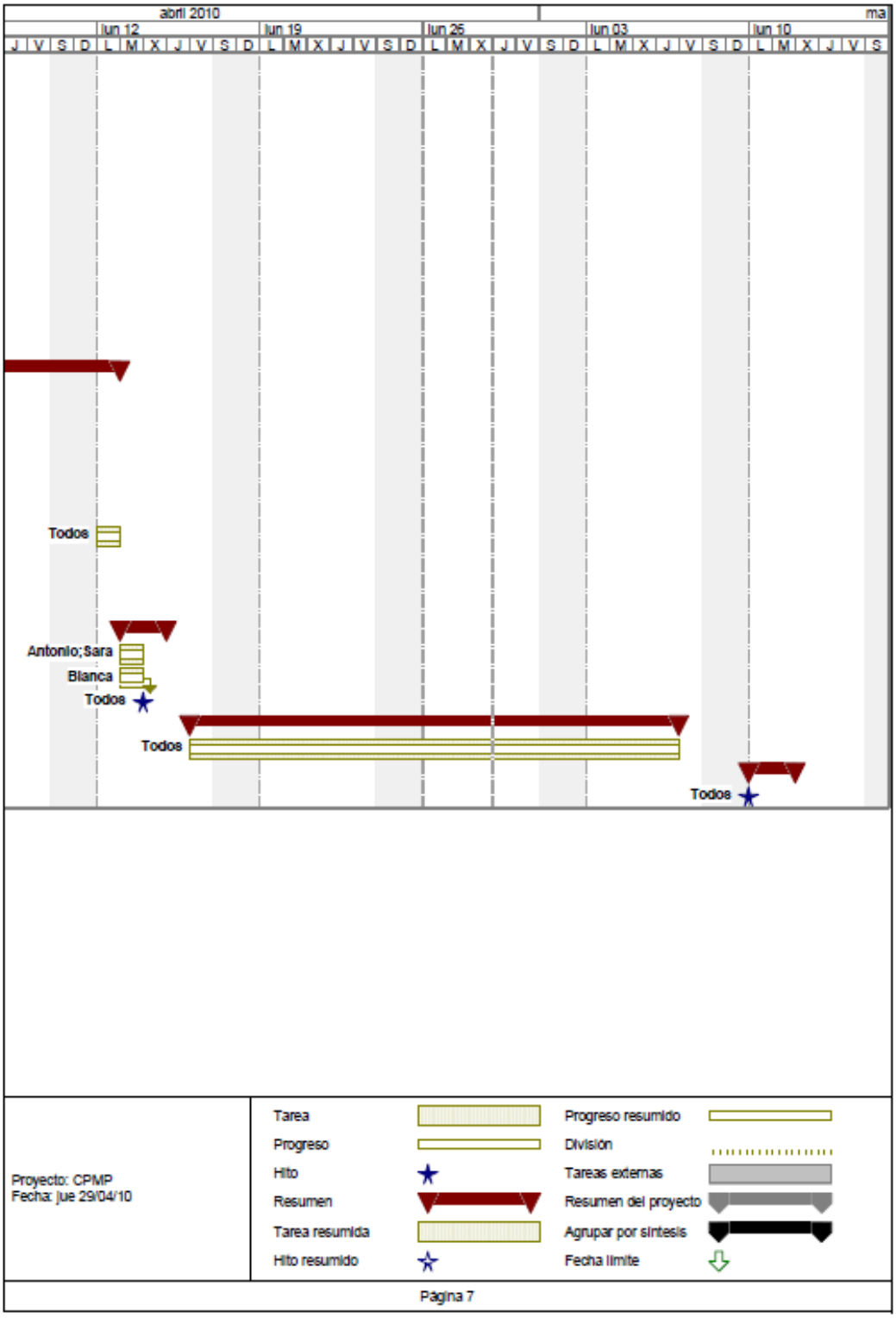

octubre 2009
lun 05
lun 12
lun 19
lun 26
lun 02
X J V S D L M X J V S D L M X J V S D L M X J V S D L M X J V
Todos
Todos
Proyecto: CPMP
Fecha: jue 29/04/10
Tarea
Progreso
Hito
Resumen
Tarea resumida
Hito resumido
Progreso resumido
División
Tareas externas
Resumen del proyecto
Agrupar por síntesis
Fecha límite
Página 2


noviembre 2009
dl
lun 09
lun 16
lun 23
lun 30
lun 07
lun
S D L M X J V S D L M X J V S D L M X J V S D L M X J V S D L M X J V S D L
Proyecto: CPMP
Fecha: jue 29/04/10
Tarea
Progreso
Hito
Resumen
Tarea resumida
Hito resumido
Progreso resumido
División
Tareas externas
Resumen del proyecto
Agrupar por síntesis
Fecha límite
Página 3


iembre 2009
enero 2010
14
lun 21
lun 28
lun 04
lun 11
lun 18
M X J V S D L M X J V S D L M X J V S D L M X J V S D L M X J V S D L M X J
Proyecto: CPMP
Fecha: jue 29/04/10
Tarea
Progreso
Hito
Resumen
Tarea resumida
Hito resumido
Progreso resumido
División
Tareas externas
Resumen del proyecto
Agrupar por síntesis
Fecha límite
Página 4


febrero 2010
lun 25
lun 01
lun 08
lun 15
lun 22
V S D L M X J V S D L M X J V S D L M X J V S D L M X J V S D
Antonio
Antonio;Sara
Sara
Sara
Todos
Antonio;Blanca
Blanca;Antonio
Blanca
Antonio
Blanca;S
Anto
Proyecto: CPMP
Fecha: jue 29/04/10
Tarea
Progreso
Hito
Resumen
Tarea resumida
Hito resumido
Progreso resumido
División
Tareas externas
Resumen del proyecto
Agrupar por síntesis
Fecha límite
Página 5


marzo 2010
lun 01
lun 08
lun 15
lun 22
lun 29
lun 05
L M X J V S D
ara
Blanca;Sara
Antonio;Sara
nio
Antonio
Proyecto: CPMP
Fecha: jue 29/04/10
Tarea
Progreso
Hito
Resumen
Tarea resumida
Hito resumido
Progreso resumido
División
Tareas externas
Resumen del proyecto
Agrupar por síntesis
Fecha límite
Página 6


abril 2010
ma
lun 12
lun 19
lun 26
lun 03
lun 10
J V S D L M X J V S D L M X J V S D L M X J V S D L M X J V S D L M X J V S
Todos
Antonio;Sara
Blanca
Todos
Todos
Todos
Proyecto: CPMP
Fecha: jue 29/04/10
Tarea
Progreso
Hito
Resumen
Tarea resumida
Hito resumido
Progreso resumido
División
Tareas externas
Resumen del proyecto
Agrupar por síntesis
Fecha límite
Página 7

APÉNDICE 7: TABLA DE USO DE RECURSOS

Tareas y recursos humanos el jue 29/04/10
CPMP

Id	Nombre del recurso	Trabajo
1	Todos	185 horas

Identificador	Nombre de tarea	Unidades	Trabajo	Retraso	Comienzo	Fin
2	Reunion Inicial	100%	5 horas	0 días	vie 25/09/09	vie 25/09/09
3	Análisis Proyecto	100%	45 horas	0 días	lun 05/10/09	vie 16/10/09
4	Diseño Proyecto	100%	25 horas	0 días	lun 19/10/09	vie 23/10/09
11	Revision NF	100%	5 horas	0 días	lun 21/12/09	lun 21/12/09
22	Ensamblaje MGC	100%	10 horas	0 días	jue 25/02/10	vie 26/02/10
34	Revision MGC+MDS	100%	5 horas	0 días	lun 12/04/10	lun 12/04/10
38	Cierre código CPMP	100%	5 horas	0 días	mié 14/04/10	mié 14/04/10
40	Memoria y documentacion	100%	75 horas	0 días	vie 16/04/10	jue 06/05/10
42	Cierre Proyecto	100%	10 horas	0 días	lun 10/05/10	mar 11/05/10

Id	Nombre del recurso	Trabajo
2	Antonio	295 horas

Identificador	Nombre de tarea	Unidades	Trabajo	Retraso	Comienzo	Fin
6	Diseño MGC	100%	25 horas	0 días	lun 26/10/09	vie 30/10/09
8	Diseño MGC(2ª vuelta)	100%	15 horas	0 días	mar 03/11/09	jue 05/11/09
9	Codificacion NF	100%	100 horas	0 días	mar 10/11/09	jue 10/12/09
12	Diseño MGC 3ª vuelta	100%	10 horas	0 días	lun 11/01/10	mar 12/01/10
13	Codificacion MD	100%	50 horas	0 días	mié 13/01/10	mar 26/01/10
17	Revision Riesgos	100%	10 horas	0 días	mié 27/01/10	jue 28/01/10
19	Diseño M GC GUI+ HELP	100%	15 horas	0 días	lun 01/02/10	mié 03/02/10
23	Revisión MGC	100%	5 horas	0 días	lun 01/03/10	lun 01/03/10
25	Revision Plan Proy	100%	10 horas	0 días	mar 02/03/10	mié 03/03/10
26	Revision Analisis	100%	10 horas	0 días	jue 04/03/10	vie 05/03/10
28	Diseño M DS	100%	15 horas	0 días	lun 08/02/10	mié 10/02/10
30	Diseño M DS 2ª vuelta	100%	20 horas	0 días	lun 15/02/10	jue 18/02/10
33	Ensamblaje M GC + M DS	100%	5 horas	0 días	jue 25/03/10	jue 25/03/10
36	Revision Planificacion	100%	5 horas	0 días	mar 13/04/10	mar 13/04/10

Tareas y recursos humanos el jue 29/04/10
CPMP

Id	Nombre del recurso	Trabajo
3	Blanca	265 horas

Identificador	Nombre de tarea	Unidades	Trabajo	Retraso	Comienzo	Fin
7	Revision Diseño MGC	100%	5 horas	0 días	lun 02/11/09	lun 02/11/09
9	Codificacion NF	100%	100 horas	0 días	mar 10/11/09	jue 10/12/09
10	Prueba NF	100%	20 horas	0 días	lun 14/12/09	jue 17/12/09
12	Diseño MGC 3ª vuelta	100%	10 horas	0 días	lun 11/01/10	mar 12/01/10
14	Prueba MD	100%	10 horas	0 días	mié 27/01/10	jue 28/01/10
15	Revision MD	100%	10 horas	0 días	lun 01/02/10	mar 02/02/10
23	Revisión MGC	100%	5 horas	0 días	lun 01/03/10	lun 01/03/10
28	Diseño M DS	100%	15 horas	0 días	lun 08/02/10	mié 10/02/10
29	Revision Diseño M DS	100%	5 horas	0 días	lun 15/02/10	lun 15/02/10
31	Codigo M DS	100%	70 horas	0 días	mar 02/03/10	lun 22/03/10
32	Prueba M DS	100%	10 horas	0 días	mar 23/03/10	mié 24/03/10
37	Cierre Ingenieria	100%	5 horas	0 días	mar 13/04/10	mar 13/04/10

Id	Nombre del recurso	Trabajo
4	Sara	275 horas

Identificador	Nombre de tarea	Unidades	Trabajo	Retraso	Comienzo	Fin
8	Diseño MGC(2ª vuelta)	100%	15 horas	0 días	mar 03/11/09	jue 05/11/09
7	Revision Diseño MGC	100%	5 horas	0 días	lun 02/11/09	lun 02/11/09
10	Prueba NF	100%	20 horas	0 días	lun 14/12/09	jue 17/12/09
13	Codificacion MD	100%	50 horas	0 días	mié 13/01/10	mar 26/01/10
14	Prueba MD	100%	10 horas	0 días	mié 27/01/10	jue 28/01/10
19	Diseño M GC GUI+ HELP	100%	15 horas	0 días	lun 01/02/10	mié 03/02/10
20	Codificacion GUI+Help	100%	50 horas	0 días	jue 04/02/10	mié 17/02/10
21	Prueba GUI + HELP	100%	20 horas	0 días	jue 18/02/10	mar 23/02/10
31	Codigo M DS	100%	70 horas	0 días	mar 02/03/10	lun 22/03/10
32	Prueba M DS	100%	10 horas	0 días	mar 23/03/10	mié 24/03/10
33	Ensamblaje M GC + M DS	100%	5 horas	0 días	jue 25/03/10	jue 25/03/10
36	Revision Planificacion	100%	5 horas	0 días	mar 13/04/10	mar 13/04/10

APÉNDICE 8: ACTAS

Acta 1

Fecha: 25-09-2009

Temas Tratados

- Se establece la necesidad de usar un almacenamiento de datos inconexos, ya que cada tipo de catálogo tiene un conjunto de campos distintos y es necesario que puedan mezclarse entre ellos. Por ejemplo, existen varios campos referentes al brillo de una estrella (Vmag, Jmag, FMag,…) y no todos los catálogos los tienen todos.
- Es necesario conocer qué campos tienen exactamente los catálogos y, con esta información, conseguir que la aplicación sea los más configurable posible. Estos datos se obtendrán del fichero readme de cada catálogo.
- Se establece la utilización del motor de base de datos MySQL, aunque se determina, como futura mejora de la aplicación, que la elección del motor sea configurable y permita al usuario elegir el tipo de base de datos (Access, Oracle,…).
- Familiarización con los atributos que definen a una estrella doble:
  - Coordenadas de la estrella más brillante de la pareja (primaria).
  - Coordenadas de la magnitud V de ambas.
  - Separación entre las dos, en segundos.
  - Ángulo que forma la estrella primaria con la estrella Polar, en grados.
  - Fecha de la medida, el año.
  - Velocidad de desplazamiento, en ms/año. Son dos magnitudes, PMRA (movimiento propio en ascensión recta) y PMDE (movimiento propio en declinación recta). Se tiene en cuenta una estimación del error cometido: EPMRA y EPMDE.

Plan de trabajo

- **Fecha entrega:** 7-10-2009

- Investigar sobre cómo convertir/importar un fichero en formato .txt a una base de datos.
- Investigar sobre el fichero readme que contiene la información de un catálogo (qué campos y parámetros incluye).
- Investigar sobre cómo tomar todo el mapa del cielo usando la aplicación VizieR sabiendo que la AR varía entre 0 y 360° y la DE, entre -90° y 90°.
- Comenzar a familiarizarnos con la aplicación VizieR y la descarga de catálogos.

Acta 2

Fecha: 7-10-2009

Temas Tratados

- Se establecen el conjunto mínimo de catálogos que debemos tener descargados para poder comprobar que la aplicación funciona correctamente: NOMAD, UCAC3 y PPMX, siendo estos últimos los más importantes.
- Se explican los problemas encontrados al descargarnos los catálogos manualmente y se llega a la conclusión de la necesidad de añadir restricciones para la descarga.

Plan de trabajo

- **Fecha entrega:** 19-10-2009
  - Desarrollar un pequeño programa que reciba los grados de separación y los de solapamiento y genere automáticamente las divisiones necesarias para descargar todo el catálogo. Es necesario tener en cuenta que debemos cubrir los dos hemisferios.
  - Descargar en nuestros equipos los catálogos antes mencionados, con algunas restricciones en sus campos para que el tamaño de los ficheros sea manejable.
    - pmRA > 70, pmRA <-70
    - pmDE > 70, pmDE <-70
    - Jmag < 17
    - Vmag < 16
    - Fmag < 22 (sólo para el UCAC3)
  - Pensar el esquema de la base de datos

Acta 3

Fecha: 19-10-2009

Temas Tratados

- División del proyecto en dos fases: gestión de catálogos y búsqueda de nuevos pares de movimiento propio común (CPMP).
- Se comienza a diseñar la interfaz de usuario, con las dos opciones anteriores.
- Se establecen las opciones iniciales del módulo de gestión de catálogos:
  - Añadir un nuevo catálogo
  - Añadir a un catálogo ya existente
  - Eliminar un catálogo
  - Visualizar el esquema de la base de datos
  - Visualizar una instancia
- Se establece el diseño de la base de datos con un diagrama Entidad-Relación

Plan de trabajo

- **Fecha entrega:** 26-10-2009
  - Comenzar la adición de los catálogos. Inicialmente se decide usar la funcionalidad de MySQL que permite importar datos desde un fichero de texto.
    - Podemos suponer que es un archivo de texto plano con los datos separados por ';'
    - Es necesario eliminar la cabecera del fichero y otros datos innecesarios para crear un fichero que pueda ser utilizado por MySQL para crear las tablas.
    - Los tipos de los campos se encuentran en la cabecera: usaremos Float, Varchar(25) e Integer.
  - Investigar cómo funciona el importar desde fichero en MySQL

- Investigar sobre el uso de un repositorio en Google Code para mantener el proyecto actualizado y sincronizado con los miembros del mismo.

Acta 4

Fecha: 26-10-2009

Temas Tratados

- Debido a que es necesario un parseo previo del fichero de entrada para obtener los campos y sus tipos, se descarta la utilización de la funcionalidad de MySQL que importa los datos desde un fichero y se opta por continuar con el parseo inicial para conseguir los datos.
- Se comienza a utilizar el repositorio creado en Google Code y se comenta su uso.
- Se propone que, al importar un catálogo nuevo, si es del mismo tipo que uno ya existente, este último se actualice con los datos del nuevo.

Plan de trabajo

- **Fecha entrega:** 2-11-2009
  - Continuar con el parseo del fichero.

Acta 5

Fecha: 2-11-2009

Temas Tratados

- Se discute sobre la posibilidad de que se importe un mismo catálogo (por ejemplo, uno de tipo NOMAD) varias veces con diferentes restricciones (es decir, que uno tenga campos que el otro no tenga). Opciones disponibles:
  - Ir añadiendo los catálogos y detectar cuando llega una parte nueva y actualizar
  - Añadir los catálogos de forma independiente y que sea el usuario el que seleccione los que se van a unir, siempre con la restricción de que tengan los mismos campos y en el mismo orden. Nos decantamos por esta segunda opción.
- Para la representación de los datos (heterogéneos) de cada catálogo, se propone utilizar una clase Estrella que contenga un array de Object con los campos del catálogo y dos campos de tipo Float para la AR y la DE.
- Se precisa que la lógica de las operaciones a implementar se realizará mediante MySQL y no con Java.

Plan de trabajo

- **Fecha entrega:** 10-11-2009
  - Completar la funcionalidad de unir dos partes de un catálogo.
  - Investigar sobre librerías de Java que permitan mostrar una tabla SQL por pantalla o en un JTable.
  - Desarrollar una rutina que tome el fichero y devuelve un objeto de la clase Estrella con los campos y sus tipos o bien un objeto de tipo Descripcion Catálogo, que será el que haga los create table.
  - Asegurarnos de que, si al parsear el fichero obtenemos uno nuevo sólo con los datos y hacemos un import desde ese fichero con MySQL, no se provocan errores cuando hay datos duplicados o se ignoran dichos errores.
- **Fecha entrega:** Por determinar
  - Ampliar la funcionalidad de unir dos catálogos para que sirva para varios a la vez.

Acta 6

Fecha: 10-11-2009

Temas Tratados

- Al no haber hecho uso del import de MySQL y generar al parsear un ArrayList de Object por cada fila del fichero (cada estrella del catálogo) con tantos Objects como campos tenga ese catálogo, se opta por crear la tabla de MySQL directamente: se hará un insert por cada línea del fichero.
- En el caso de que no se consiga dominar el uso de la librería que muestra una tabla de MySQL, se propone mostrar un JTable de X filas junto con dos botones Anterior y Siguiente.

Plan de trabajo

- **Fecha entrega:** 23-11-2009
  - Metodología a seguir cuando el usuario quiere añadir un catálogo:
    - Se pide al usuario un nombre para el catálogo: será la clave de la tabla de catálogos.
    - Se parsea el fichero y se crean las tablas
    - Se almacena esta tabla en la "tabla de tablas" y se añaden los datos.
    - Al finalizar el proceso se debe mostrar el número de filas añadidas y descartadas.
    - En algún punto intermedio entre el parseo la creación de las tablas, se convierte el objeto de tipo Object en un objeto del tipo correspondiente (Float, Varchar(25) o Integer).
  - Configurar la aplicación para que, al iniciar, pida al usuario un identificador y su contraseña.
  - Asegurarnos de que la conexión y desconexión a la base de datos se hace al iniciar y finalizar la aplicación, para optimizar el rendimiento.
  - Es necesario añadir un identificador numérico a cada campo.
  - Poder mostrar los catálogos ya añadidos.

Acta 7

Fecha: 23-11-2009

Temas Tratados

- Se comienza a definir la segunda parte de la aplicación: la creación de un lenguaje de consulta de tipo script. Algunas de las instrucciones serán:
  - Renombrar un catálogo
  - Combinar dos catálogos
  - Obtener pares cercanos (se pedirá al usuario un criterio de proximidad en segundos)

Plan de trabajo

- **Fecha entrega:** 30-11-2009
  - Visualizar la tabla de campos para un catálogo seleccionado.
  - Para hacer la funcionalidad Join, asegurarnos de que los datos están ordenados y comparar los resultados y el rendimiento obtenidos al usar la comparación de uno en uno o en bloques.

Acta 8

Fecha: 30-11-2009

Temas Tratados

- Cambio del nombre de los menús de la GUI
- No se necesita pedir al usuario el tamaño inicial de la visualización de catálogos: se establece por defecto.

Plan de trabajo

- **Fecha entrega:** 21-12-2009
  - Al finalizar el Join debe mostrarse el número de pares encontrados.
  - Metodología para realizar el Join: Divide y Vencerás
  - El usuario es el que debe definir el tamaño del rango
  - Estudiar cuánto tiempo se tarda en hacer los deletes de la tabla temporal

Acta 9

Fecha: 21-12-2009

Temas Tratados

- Se establece que todas las operaciones deben crear una tabla nueva.
- Se define el resto de las operaciones disponibles en los scripts: rename, merge, join, filter, distance, minus, attribute y newAttribute.

Plan de trabajo

- Fecha entrega: 18-01-2010
    - Implementar una operación Minus que compare dos tablas y devuelva aquellas entradas de la primera que no estén en la segunda. Dicha operación servirá para filtrar los pares ya descubiertos utilizando el catálogo WDS.

Acta 10

Fecha: 18-01-2010

Temas Tratados

- Se establece que la aplicación debe disponer de una ayuda auto-explicativa para que el usuario pueda manejarla sin necesidad de ningún otro documento.
- Se establece que sea el usuario el que se asegure de que los catálogos a importar:
  - Dispongan de una cabecera adecuada.
  - Contengan los campos obligatorios RAJ2000 y DEJ2000.
  - No incluyan campos duplicados.
- Se presenta el módulo inicial de gestión de scripts y se añade una nueva funcionalidad: la posibilidad de que el usuario pueda crear sus propias funciones y utilizarlas en diferentes scripts

Plan de trabajo

- Fecha entrega: 25-01-2010
  - Implementar la funcionalidad anterior de modo que sea el usuario el que decida cuáles de los parámetros serán de entrada y cuáles de salida.

Acta 11

Fecha: 25-01-2010

Temas Tratados

- Se recuerda que todas las operaciones del módulo de scripts deben funcionar para tablas de pares y tablas individuales.
- Se propone la exportación de los catálogos existentes en la aplicación a ficheros de texto.

Plan de trabajo

- Fecha entrega: 15-02-2010
  - Asegurarnos de que las operaciones ya implementadas funcionan también con tablas de pares.
  - Modificar el menú principal de la GUI para que sea más intuitivo.
  - Implementar el export.

Acta 12

Fecha: 15-02-2010

Temas Tratados

- Se elimina la obligación de que los catálogos a importar deban ser archivos de texto con extensión .tsv
- Se añade a la visualización de los catálogos, en el apartado de *General Settings*, el número de estrellas de cada catálogo.
- Se suministra la operación export y se propone que también cree una cabecera válida para poder volver a importarlo correctamente.

Plan de trabajo

- Fecha entrega: 01-03-2010
  - Generar una cabecera válida al exportar un catálogo.
  - Probar distintos casos en los scripts para detectar posibles fallos en el análisis sintáctico.

Acta 13

Fecha: 01-03-2010

Temas Tratados

- Se añade una nueva funcionalidad al importar un catálogo: importar un catálogo creando una cabecera válida. Dicha cabecera se creará preguntando al usuario (rellenando un formulario).
- Se propone la integración de la aplicación online Aladin para poder visualizar las estrellas a través de las placas fotográficas disponibles en dicha aplicación.

Plan de trabajo

- Fecha entrega: 08-03-2010
  - Implementar la funcionalidad para importar un catálogo creando la cabecera.
  - Arreglar los errores cometidos al desarrollar la operación export.
  - Hacer que, al seleccionar la ayuda en cada sección de la aplicación, la ventana de ayuda muestre el apartado referido a dicha sección.
  - Integrar Aladin en la aplicación.

Acta 14

Fecha: 08-03-2010

Temas Tratados

- Implementar la opción de eliminar campos de un catálogo mediante click derecho del ratón

Plan de trabajo

- Fecha entrega: 15-03-2010
    - Continuar con la creación de una cabecera al importar el catálogo.

Acta 15

Fecha: 15-03-2010

Temas Tratados

- Se expone la opción de sobrescribir los resultados al ejecutar un script, de modo que la ejecución sucesiva del mismo script no genere errores de tipo *tabla existente*.

Plan de trabajo

- Fecha entrega: 22-03-2010
    - Implementar la opción anterior dando al usuario la posibilidad de elegir, en tiempo de ejecución, sobrescribir o no los resultados.

BIBLIOGRAFÍA

[AD99] Allende Prieto, C; Dambert D.L.; 1999, *"Fundamental parameters of nearby stars from the comparison with evolutionary calculations: masses, radii and effective temperatures"*. Astronomy and Astrophysics, 352, p.555-562 (1999).

[BFBEGLOWB00] Bonnarel, F.; Fernique, P.; Bienaymé, O.; Egret, D.; Genova, F.; Louys, M.; Ochsenbein, F.; Wenger, M.; Bartlett, J. G.; 2000, "*The ALADIN interactive sky atlas. A reference tool for identification of astronomical sources*", Astronomy and Astrophysics Supplement, 143, p.33-40.

[B91] *Boehm, B.W. "Software Risk Management: Principles and Practices". IEEE Software 1991*".

[C10] Caballero, R., "*Finding New Common Proper-Motion Binaries by Data Mining".* Journal of Double Star Observations, Volume 5, Number 3. Pages 156-167. 2009.

Disponible en Web:

http://www.jdso.org/volume5/number3/Caballero2.pdf

[G04] Greaves, J., 2004, "New Northern hemisphere common proper-motion pairs". Monthly Notices of the Royal Astronomical Society 355, 585-590.

[H86] Halbwachs, J.L., *Common proper motions stars in AGK3*.

Disponible en web: VizieR On-line Data Catalog: I/121. Originally published in: 1986A+AS...66..131H

[M81] Mantei, M. "The effect of Programming Team Structures on Programming Task". CACM Marzo 1981.

[P05] Pressman, Roger S., *Ingeniería del Software: Un enfoque práctico*. Editorial McGraw Hill; Sexta edición, 2005.

[R99] Ridpath, Ian. Diccionario de Astronomía. Editorial Oxford-Complutense; 1999.

[S05] Sommerville, lan., *Ingeniería del Software*. Editorial Addison-Wesley Iberoamericana España; Primera edición, 2005.

SH94654 IEEE Computer Society. *IEEE Recommended Practice for Software Requirements Specifications* [en línea]. [20 October 1998]. Disponible en Web: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=720574

SQAS22.01.00-2002 Department of Energy Quality Managers Software Quality Assurance Subcommittee. *Software Quality Assurance Control of Existing Systems* [en línea]. [September 2002]. Disponible en Web: http://cio.energy.gov/Sftwr_Qlty_Ass-22_01_00-2002.pdf